UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

TRABAJO FIN DE MÁSTER

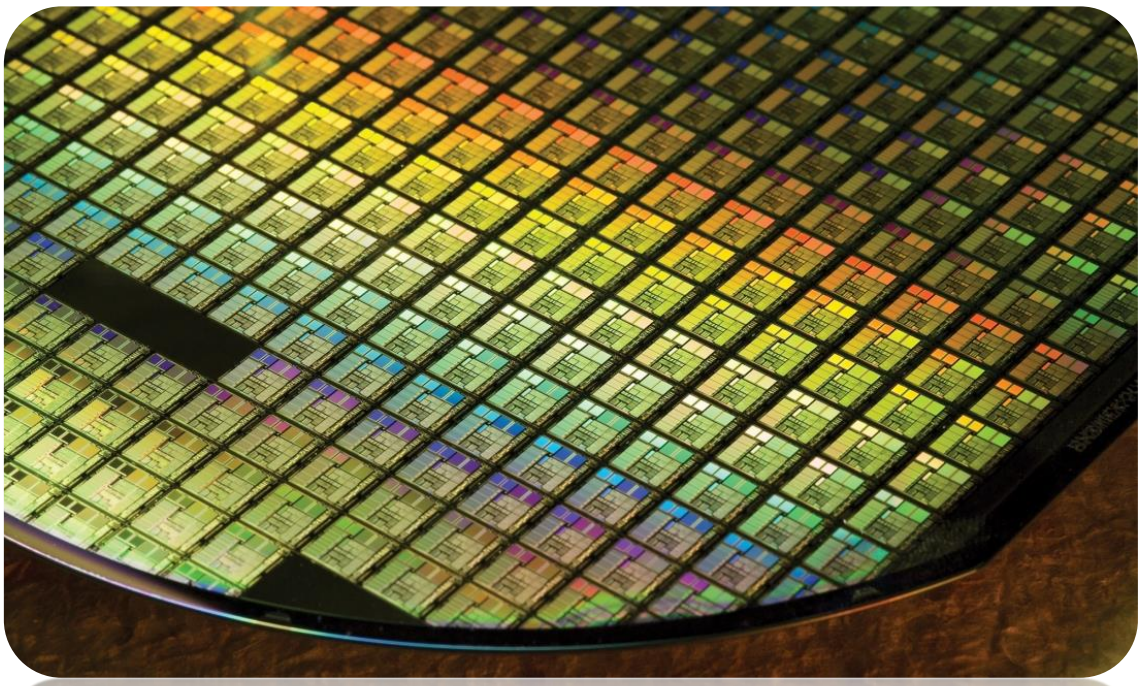
Analyzing Part Lifecycle inside a manufacturer of chip-making equipment

Máster Universitario en Ingeniería Informática

Autor: Carrascosa García, Fernando

Director: Engels, Loek
Co-Director: Antonissen, Ronald
Ponente: Sutter Capristo, Gustavo

Septiembre 2017



Wafer
Source: TechSpot

ANALYZING PART LIFECYCLE INSIDE A MANUFACTURER OF CHIP-MAKING EQUIPMENT

Fernando Carrascosa García
Madrid – Spain

September 2017

“All we have to decide is what to do with the time that is given to us.”

J.R.R. Tolkien, The Fellowship of the Ring

Resumen

El presente documento trata sobre el desarrollo de una aplicación software destinada al Análisis del Ciclo de Vida de componentes dentro de un fabricante de maquinaria productora de chips electrónicos. La intención del mismo es exponer de forma detallada los objetivos del proyecto, el contexto en el cual queda enmarcado y las metodologías seguidas para su consecución. De la misma manera se exponen también el plan de trabajo y la evolución del software a través de las distintas fases de desarrollo.

La aplicación software presentada, dada su complejidad y carácter profesional, fue considerada como óptima para ser expuesta como Trabajo Fin de Master (TFM) por parte del alumno autor de este documento, dentro del programa "Master en Ingeniería Informática" impartido en la Escuela Politécnica Superior de la Universidad Autónoma de Madrid. Haber encontrado un nexo de unión entre un proyecto de carácter académico y un proyecto de carácter profesional se considera un logro en este caso. El software resultante del proceso de desarrollo expuesto en este documento es una herramienta valiosa que será utilizada para beneficio de una compañía real.

Se considera también de especial relevancia la demostración de cómo es posible el desarrollo de una herramienta software siguiendo una metodología ágil, con un conocimiento a priori limitado y unas restricciones de tiempo y presupuesto. En un primer momento, tanto usuario como desarrollador conocían el objetivo final pero desconocían el camino para llegar hasta él. El carácter mismo de las metodologías ágiles ha potenciado la comunicación entre desarrollador y usuario final, y fue esa colaboración activa entre las dos partes la que ha llevado a este proyecto a buen término. Por último se considera igualmente relevante el estudio realizado acerca de un tema de creciente importancia dentro de varios sectores industriales como es el análisis de ciclo de vida de producto.

Palabras clave

Ciclo de Vida del Producto, Metodologías Ágiles de Desarrollo Software.

Abstract

This document deals with the development of a software application for the Lifecycle Analysis of components in a manufacturer of chip-making equipment. The aim of the document is to show in a detailed manner the project objectives, the context in which it is framed, and the methodologies followed for its execution. The working plan and the application evolution along development phases are described as well.

Due to its complexity and professional use, the application was considered optimal to be presented by the author, student of Master in Computer Engineering in the "Escuela Politécnica Superior de la Universidad Autónoma de Madrid", as his End of Master Thesis (EMT). The opportunity to find a link between an academic work and a professional project is indeed a unique achievement. The outcome of the development process described in this document is a valuable tool used for the benefit of a real company.

Showing how a software tool can be developed following an agile methodology, with limited initial knowledge and severe constraints both on time and budget, is equally considered a relevant achievement. In the first moment, the end user and the developer knew the final objective, but not how to realize it. The nature of agile methods has boosted the communication between them, and this proactive collaboration has led the project to the right end. As Product Lifecycle Analysis has become of increased importance in various sectors, this study can be considered as a relevant contribution to current industrial research.

Keywords

Product Lifecycle, Agile Software Development Methodologies.

Contents

Tables

Figures

Glossary

Abbreviations used across the document:

Abbreviations	Description
12NC	12-digit Numeric Code
BOL	Beginning Of Life
EOL	End Of Life
EMT	End of Master Thesis
EN	Equipment Number
FDD	Feature Driven Development
IID	Iterative and Incremental Development
MOL	Middle Of Life
PDM	Product Data Management
PDR	Product Deliverable Report
PIP	Part Info Page
PL	Product Lifecycle
PLM	Product Lifecycle Management
PM	Product Management
PO	Purchase Order
SCE	Supply Chain Engineering
SNP	Serial Number Profile
TCD	Test Coverage Document
TSV	Tab-Separated Values
TPD	Technical Product Documentation
UCC	Under Change Control
VQI	Volume Qualification Indicator
WBS	Work Breakdown Structure
WIP	Work In Progress

Table 1: Glossary

1 Introduction

This End of Master Thesis Project is located inside the context of **Horus** (pseudonym), an information hot-spot for a large supplier of photolithography systems, hereinafter the sponsor company. The aim of Horus is to provide a single source of easily accessible information, in order to support and organize the company processes. The Horus team consists of a group of software engineers who involve the key-users in the development process through constant and fluent communication. Following agile methodologies, the solutions are flexible and quickly delivered.

Fernando Carrascosa joined the team in March 2016. Since then, he has worked in several modules of the system, learning from other team members and understanding the intricacies of this long road project which evolves constantly. The opportunity of doing an End of Master Thesis Project appeared in October 2016 when new requirements arose and Fernando´s experience was mature enough to take the responsibility to analyze, design and develop a new key piece of the system.

The information used by the sponsor company is ultimately related with elements called **parts**. A part is an item produced or bought by the company. In some cases they could be simple items, i.e. special screws, while in others they could be assemblies composed by aggregation of simple items, e.g. a set of lenses positioned inside a case with standard fasteners to redirect a laser ray. Therefore, parts could be simple or compositional items and a set of parts may in their turn form a higher order part. Depending on their complexity, parts are managed in different ways. Part management is an activity shared among many industries, for example the automotive. Figure 1 shows a Mustang GT500 engine exploded as an example of a product made from many components with different complexities.

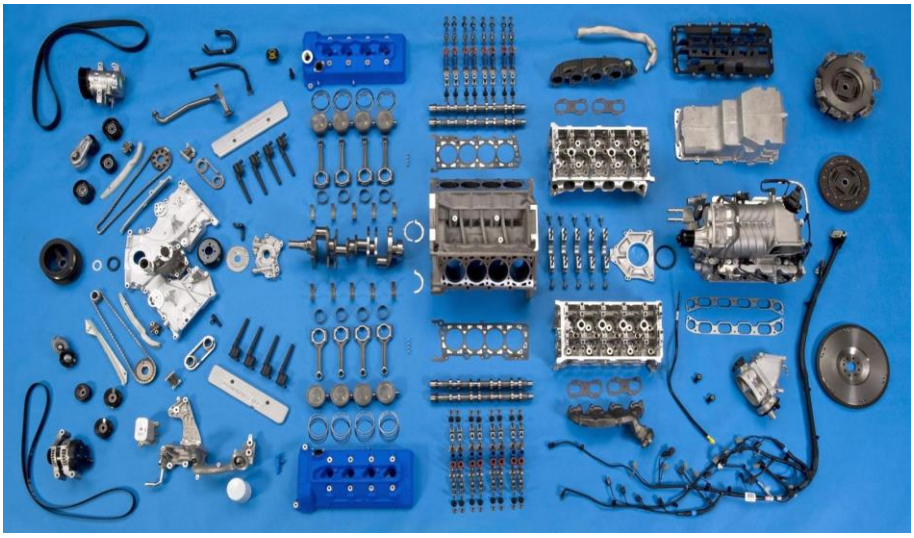


Figure 1: Mustang GT500 engine exploded
Source: Mustang 360º - 2013 Ford Shelby GT500 Trinity 5.8L V8

All the parts are univocally identified by a numeric code named **12NC** (12-digit Numeric Code). In most cases parts have a long development process from the moment they are created until their end of life. The life of a part usually starts with a designing phase. There are parts that stay several months in the phase of designing until they actually exist. This implies that parts do not necessarily have to be tangible. Thus, a 12NC could refer to something that is only partially designed.

In order to meet customer expectations, the sponsor company is continuously driving, designing and developing new and better machines. This means that the sponsor company is designing lots of new parts on a daily basis. Each and every of these parts need to be able to function under certain production circumstances once delivered to the customer. For that reason, they have to mature from early design prototypes to a fully functional part. Industrialization is the key to make this happen. The processes and sub-processes that determine the life of a part are in accordance with the well-known Product Lifecycle which is common to almost every industrial field.

The continuous improvement process does not end once the parts are in use. Issues can occur and have to be solved, or internal actions like risks assessment or simple technical evolution could result in change proposals to be implemented in the field. All these maintenance activities are usually documented and stored in a huge variety of data systems. The existence of these multiple data sources usually hampers a holistic view on the entire lifecycle of the part. This project aims to facilitate this task within Product Lifecycle analysis.

The goal of this project was to create an application that shows the entire lifecycle of a part in the sponsor company. First of all, it was necessary to pay attention to the key attributes, deliverables and issues which are important for the engineers to assess whether the industrialization of a part has been successful or not. Then, the source systems where the part data is stored were investigated, and a link to them was created. Finally, the information was reported via a Horus user interface. The developed application has received the name **12NC Timeline** and this term will be used in the rest of the document.

1.1 Motivation

The student has participated in a software development team known by its ability to deliver quality software. This is partially achieved by putting in practice some of the principles of agile software development methodologies. One of the advantages of this work model is the capacity of getting things done. There is a maxim in the team, “better to release something that works than release nothing at all”. This is a really important concept in the application that has been developed. At the beginning, requirements were vague and the stakeholder only had an idea of what he wanted. Then, working with small iterations helped defining new requirements. The application was shaped by user feedback. At the end, **12NC Timeline** is a good example of how agile methodologies support an effective software development.

Regarding the project functionality, the application developed is already gaining importance inside the sponsor company. The idea of having a single trustable point to consult the whole lifecycle of a part is very attractive, and at the moment the application is already being used by a relevant number of engineers.

From the academic point of view, the Master in Computer Engineering given by "Universidad Autónoma de Madrid" is posed as a master for students who want to gain a deeper knowledge on several Computer Science fields. Those students are going to face sooner or later the real challenges that usually appear in professional environments. Therefore, the opportunity to have developed a project like this inside a professional environment is particularly unique. It represents a significant step in the professional career of the student.

1.2 Objectives

The main goal of this project was to create an application that would show the entire lifecycle of a part inside the sponsor company. This referred to every step the part was facing from its creation to the end of its life. The application included a single-entry page in which the engineer would be able to find the most important deliverables, attributes and issues of a part. This would allow the engineer to easily assess whether the industrialization of that part has been successful or not. The partial objectives were the following:

- Determining the key-attributes, deliverables and issues that are important for the engineer in order to track the part industrialization process. This point focused on the requirements analysis. The sponsor company stakeholders and other users of the application suggested additional requirements once the project was under development. Thus, an iterative process was followed to determine those key-attributes.

- Analyzing in which data source the information could be found. The sponsor company counts with a wide range of data sources where the relevant information about the company processes is stored. Horus is one of these data sources and the most preferred one to perform this analysis task. Nevertheless, it was known that some information regarding these key attributes was not available in Horus. Stakeholders had to decide what to do in that case.

- Developing a Horus user interface for reporting the information. The first approach was to list all the attributes in a chronological order following a table structure. In order to display this complex table, some software libraries present in Horus project were used. They will be presented in section 3.2. An agile software development model (see section 3.1) was applied to carry out the development activities. To be *agile* was a must in this project.

The partial objectives were achieved in each of the several incremental deliverables that shaped the final product. Therefore, some value was added in each iterative step, and the completion of the partial objectives reached the main one: to analyze the part lifecycle inside the sponsor company using Horus framework.

1.3 Document structure

The further course of this paper is divided in several chapters as follows:

- **Context:** Chapter 2 offers a more detailed description on the Project Context, including additional information about the sponsor company business (photolithography), the suite Horus, and an introduction to the topic of Product Lifecycle Management (PLM), whose improvement is the Project leitmotiv.

- **Methodology:** Chapter 3 contains a general description of Software Development Methods comparing the "Classic" models with the "Agile" ones. The most significant methodologies for the latter are presented (Kanban and Feature Driven Development), as well as the set of tools used along the project both for development and planning. Nine tools are described with a justification for their usage in the project.

- **Work plan:** Chapter 4 contains the Project Work Plan, including the basic Work Breakdown Structure (WBS) and the initial work load assessment, as well as a complete and detailed definition in an intermediate stage just for showing how the planning tool (JIRA) and some of the methodologies (i.e. Kanban) described in previous chapter have been used.

- **Development:** Chapter 5 elaborates on the development process, which was divided in three phases according with the three deliverables that were presented to the user. The challenges for each of the phases are explained in detail and some images are included in the result subsections to better show the final functionality.

- **Conclusions:** Chapter 6 contains a short summary of the work presented. The achievement of the partial and general objectives is also analyzed, together with future work and possible improvements.

- **Appendix:** Additional information not included in the main part of the document due to its extension, but interesting as a reference, is offered as an Appendix out of chapter numbering. The detailed Project Planning has been included in this section.

2 Context

Some details about the context of the project are offered to the reader in this section. First, the company which is going to benefit from this project outcome software will be briefly introduced; then, some concepts about Product Lifecycle Management and how it is related with this project are explained; and finally, the team where this project was developed will be presented;. To have a proper context in mind will help to understand further sections of the document.

2.1 Photolithography

The word lithography comes from the Greek *lithos* (λίθος), meaning "stone", and *grapho* (*γράφω*), meaning "to write". Thus, lithography can be literally translated as "*writing on stones"*. In the case of semiconductor lithography (also called photolithography) the stones are silicon wafers and the patterns are written with a light sensitive polymer called a photoresist [1]. Light (UV) is the key element to print the final design of the integrated circuit into the semiconductor, but several physical and chemical reactions take place during the fabrication. The sequence is shown in Figure 2.

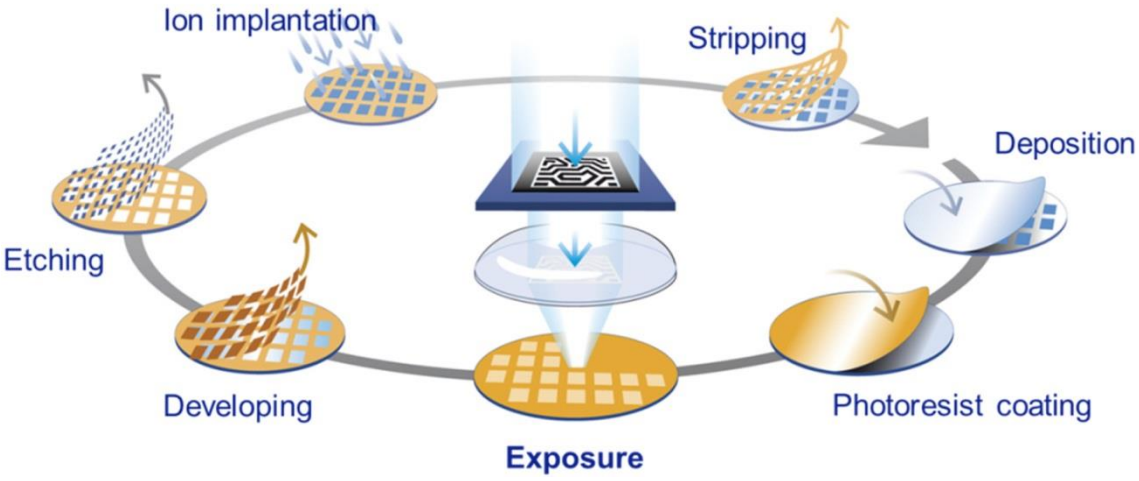


Figure 2: The semiconductor manufacturing process

Source: IOPscience [2]

The sponsor company, as stated in the Introduction, is one of the leading manufacturers of chip-making equipment and a large supplier of photolithographic systems. The machines manufactured by it are used for the production of integrated circuits and computer chips, such as CPUs, DRAM memories, etc.

2.2 Product Lifecycle Management

For the sponsor company, an example of a capital goods industrial company, the core of its operations is the Product Management (PM) from the conceptual phase to the end of life. The management activity comprises several phases: design, procurement, manufacturing, assembly, testing, configuration control through all the steps, and disposal when needed.

Those phases make up the Product Lifecycle (PL), a topic which has been studied in the industry for some decades [3]. A recent definition is given from [4], who regards PL as a flow: *"(...) the product lifecycle consists of three main phases: the Beginning Of Life (BOL) including requirements, design, industrialization and production; the Middle Of Life (MOL) including logistics, use, and maintenance; and End Of Life (EOL) including reverse logistics, remanufacturing, reuse, recycle, and disposal"*. Or in a more graphic manner, the one below:

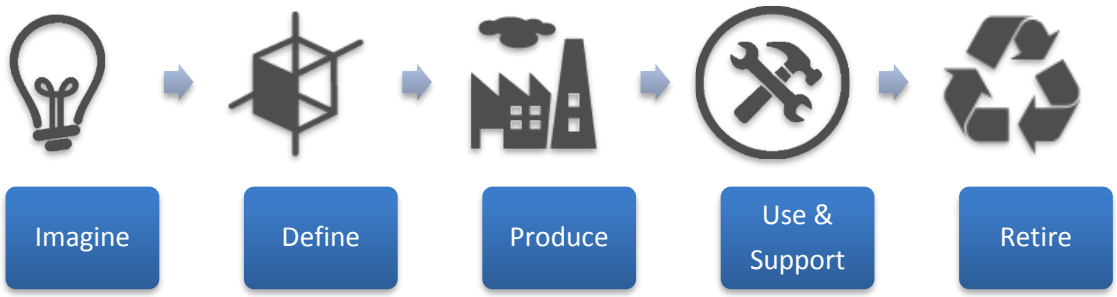


Figure 3: The five general phases of the Product Lifecycle
Source: Own representation based on [4]

The importance of this cycle lies in the capacity of a company to have a clear picture of what are the steps its products are going through, and the power to make reasonable predictions about them. Driving improvements on a process requires knowing the process flow. The activity of managing this cycle is called Product Lifecycle Management (PLM).

Stark's definition of PLM is one of the most accurate in the bibliography: *"Product Lifecycle Management (PLM) is the business activity of managing, in the most effective way, a company's products all the way across their lifecycles; from the very first idea for a product all the way through until it is retired and disposed of."* [5]. In this context, PLM is considered a strategic business approach that includes people, technology, information and processes.

For products with tens of thousands of parts, coming from hundreds of sources (external suppliers, in-house manufacturing, etc.) as described in section 2.1, and evolving in a continuous improvement context, changes occur very often. New data appear continuously and their management is a real challenge. In order to integrate all these elements, this activity is often supported by an aggregation of software tools, a "PLM system" which *"applies a set of business solutions to support the collaborative*

creation, management, dissemination, and the use of product definition information across the extended enterprise from concept to end-of-life" [4].
The software tool, among the ones mentioned above, that is used to face this challenge is the Product Data Management (PDM) system. It is one of the key elements of the PLM environment because *"It can manage all the product data created and used (...) providing the right information [at] the right time throughout the product lifecycle (...) The PDM application gets this strategic resource under control, making it available, whenever it's needed, wherever it's needed, by whoever needs it"* [5]. At this point, some questions arise naturally: How does the sponsor company manage product data? In what manner are these data recorded? What are the key interfaces? The paragraphs below and the next section answer these questions by describing the PDM system of Horus

Industrialization and Product Data collection start in the early design phase, where product specifications are translated into part specifications. These on their turn are translated into part requirements. These requirements are translated into Technical Product Documentation (TPD), which describes how a part should be, typically including material specifications, dimensions and tolerances, surface treatments, etc. This TPD is delivered to a chosen supplier who, in turn, manufactures the part according to the TPD.

Regarding the delivery process, a representative from the sponsor company and the supplier sign a Test Coverage Document (TCD) that includes the results of all qualification tests which the part was subject to. The objective of the TCD is to prevent issues with the part during its later use, although unfortunately they can still happen. The part is then delivered to the manufacturer where it is assembled into the physical product. This whole process is infused with many risks. A thorough risk assessment and mitigation process is applied all along these lifecycle steps of a new part.

The classical PLM system is a closed loop among design, industrialization and production [4]. This scheme is valid "in house" where data flow is relatively easy between company departments, but it does not cover the Extended Enterprise context, taking into account external agents like customers or suppliers. After delivery from the supplier and during the usage of a part, many things can still go wrong, leading to malfunction. In such an instance many feedback loops are possible (see Figure 4). A document called "Material Notification" (MN) is created to manage these disruptions. If a part manufactured by a supplier deviates from a particular specification (as described in the TPD) the supplier can request leniency by the sponsor company, so that he would still be able to deliver. This is done via a "Deviation Notification" (DN) process in which the supplier and the sponsor company review the deviation and agree on either use the part as given, rework or scrap it.

Once the final product has been tested, it is delivered to a customer for use. During the usage by the customer a part might fail. Several activities based on the risk assessment done by the sponsor company are carried out to prevent issues once the part is in use. The activities are often documented and the issues are registered as MNs or DNs in order to analyze the root causes. However, these pieces of information are stored in multiple data sources. This increases difficulty for an engineer to get a holistic understanding of the entire lifecycle of the part, or to answer key questions like:

Did the risk mitigating activities result in fewer issues? Do we find issues on parts that we deemed low risk? Finding the answers to this kind of questions could be a hard work, due to the existence of several source systems.

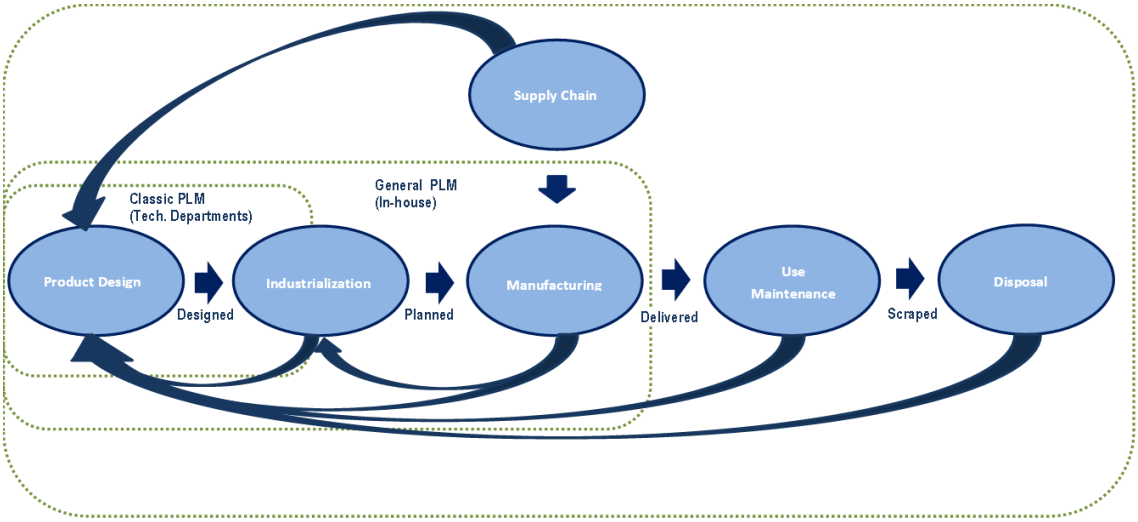


Figure 4: The Information Feedback Closed Loops in Extended Enterprise.

Modified from [4]

PDM is an activity especially relevant in technological companies producing complex products. The amount of attributes, documents and indicators related with a part lifecycle could be huge and its adequate management really critical.

The next section describes the PDM system in place for the sponsor company, with some of the limitations for PDM functionalities, caused by the situation of data being spread across several systems and the need for a tool like **12NC Timeline**. In the Development section, where the software parts are explained, some of the Product Lifecycle Management elements will be presented with more details.

2.3 Horus

Horus is a web-based suite of applications supporting the Development and Engineering processes in the sponsor company. It delivers information to developers and managers in such a way that it is easy to use. This information is coming from multiple sources such as SAP, TC and other data management systems. Horus contains a PDM system in accordance with the definition of [4] and includes most of the typical PDM system capacities. The reason to develop **12NC Timeline** is basically to improve the PLM features within Horus.

Horus adheres to agile software development model. In practical terms that means the customer is involved into the development process from the start. The customer's feedback is indeed the motor which keeps the development running. In addition, a short iterative process (weeks) is preferred rather than setting requirements that will be implemented in the long term (months).

Then, releases in Horus are planned weekly and the team adheres to an agile software development methodology based on Kanban and FDD, coming in some cases from fields not related to the software industry (see section 3.1.3). In this short iterative process, it is better to start with a minimum viable solution and to build functionality on top of it. Therefore, there is a clear focus on delivering in Horus team. That is a principle followed in **12NC Timeline** application.

Horus is successfully running twelve different applications concurrently under the same development structure. This End of Master Thesis Project has been addressed as one of them, making the student the responsible and single developer. Daily stand-ups and weekly progress meetings took place inside Horus as part of the agile model followed and the student has participated on them in order to share his progress.

Horus application functionality is based on importing, showing and exporting information. Figure 5 shows the system architecture. There are four distinguishable elements in it:

- First, there is a set of **importers** that retrieves information from various data sources. The most relevant ones from Horus perspective are SAP and TC. SAP is an Enterprise Resource Planning (ERP) software to manage business operations and relations with customers. Horus obtains information from SAP about parts in production, equipment hardware and notifications. TC is a software suite for Product Lifecycle Management (PLM). Horus obtains from TC mainly information about the part designs, bill of materials and several types of documents with their connections: technical designs, test coverage, etc.

- Second, there is a set of **updaters** that work together with the importers and keep the information up to date. This is because sometimes the information imported needs to be transformed in order to be useful, so the updaters can do that work as a different task and speed up the import mechanism.

- Third, there is a set of **exporters** that prepare the data to be dumped into external files accessible by the customer. These files are usually Excel format and give the user more freedom to play with the data. On top of the architecture schema, the Horus suite of web applications is located.

- Applications inside Horus perform as **reporters**, presenting the information from Horus database to the user in an easily accessible way. They also allow the user to enter data and to manage the information stored, so their capabilities are more than just show data.

The first three elements in the Horus architecture (importers, updaters and exporters) are considered as the back-end part of the system, while Horus reports are considered as the front-end. The interesting thing for a Horus team member is that he performs as a developer full-stack, so he should have a proper knowledge of the complete system. Developing a Horus application usually means doing some work in several parts of the system. This was obviously the case for **12NC Timeline**.

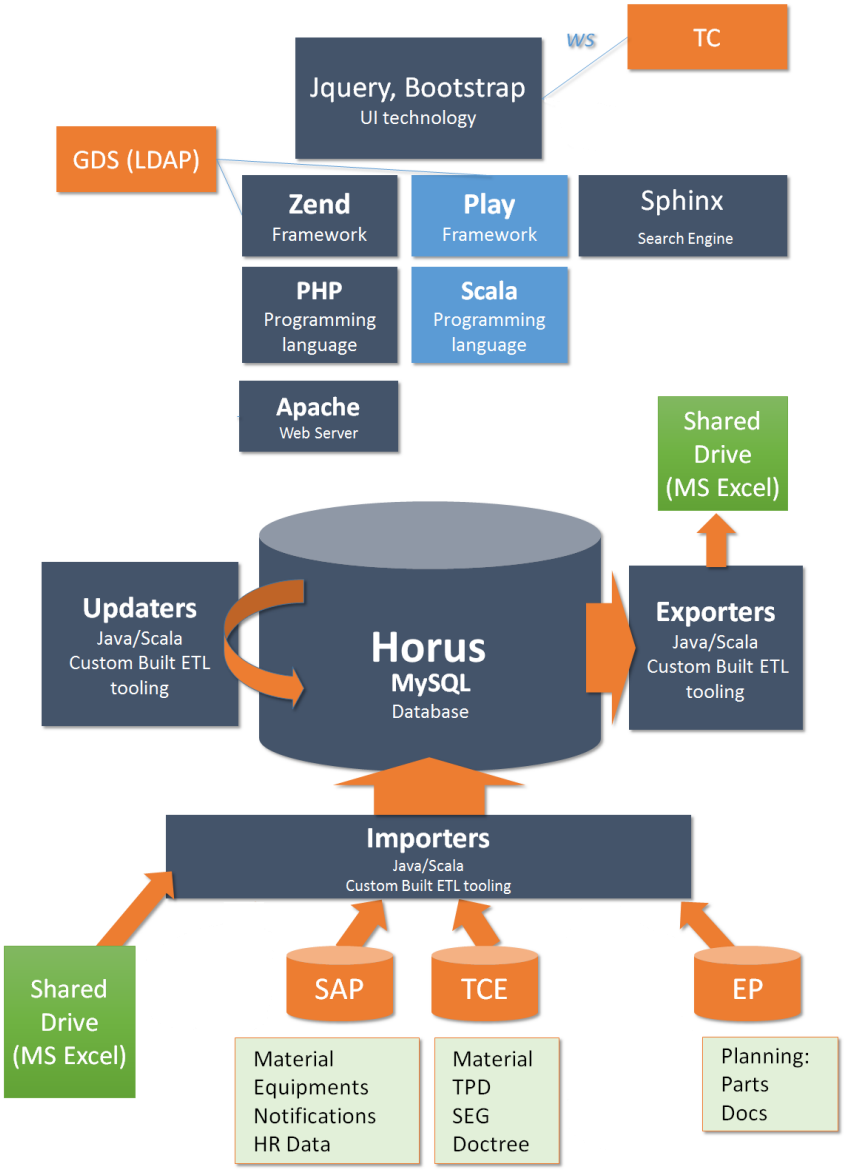


Figure 5: Horus architecture

Source: Own representation based on internal documentation

Horus is a solution only being used inside the sponsor company. It is well established and has a remarkable impact on process management of the sponsor company.

3 Methodology

It is important to make a distinction between the concepts of software "process model" and software "development methodology". A software process model is "*a simplified representation of the software process*" [6]; the term model represents a set of principles under which the software development is taking place. A software development methodology is a specific way of conducting a software project; it refers to the specific tools or steps the developer should take in order to build the software.

In this section, details about the model and methodologies used for the achievement of this project are provided. As discussed in the objectives section, this End of Master Thesis Project is focused on the development of a complete application. To achieve it, an agile software development model was followed. This means, among others, incremental gathering requirements, adaptive planning, evolutionary development and constant customer feedback. Inside that model, two different methodologies were used to build the software, Kanban and FDD. Details about both methodologies will be given together with the specific method application in Horus team and in this project.

3.1 Agile software development

The agile software development model is one of the biggest trends in the software development sector nowadays [7]. It was conceived to replace the traditional models such as waterfall or spiral model. Nevertheless, the advantages and disadvantages of these previous models should be considered to understand why the software development industry has chosen agile in the last years.

3.1.1 Classic Model Evolution

Between the seventies and the eighties, digital technology was becoming the center of the worldwide economy and there was an increasing necessity of larger and more complex software programs [8]. A very structured and rigid methodology for software development was being established at that time. It was commonly known by the name "waterfall model". The waterfall model follows a sequential process divided in several phases, being those: requirement analysis, design, implementation, testing and maintenance. In Figure 6, a graphical representation of this model is shown. The model has some important advantages:

- It enforces discipline in the development process because there are defined start and end points for every phase. Deadlines have to be accomplished and the progress can be checked at the end of each phase.
- The abundance of technical specification documents facilitates sometimes the knowledge transfer when a new developer joins the team.

- There is a strong focus on requirements definition and design. This ensures that customer expectations are met at least in the moment when the requirements were defined.

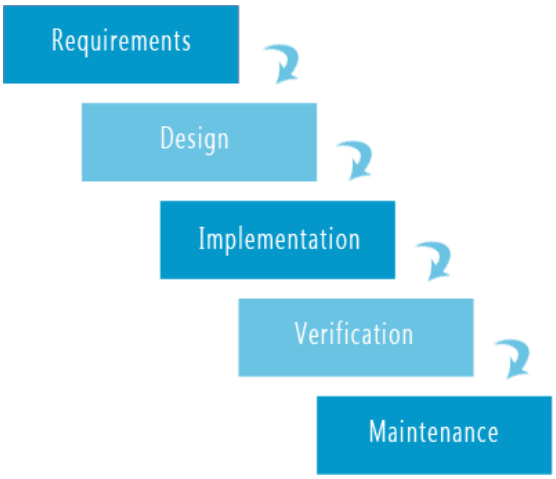


Figure 6: Waterfall model.

Source: Oxagile [9]

However, the last advantage presented leads equally to the main problem of the waterfall model: in the real world, customers do not know exactly what they want up-front most of the times and it is almost impossible to write a set of requirements that will not suffer any adjustment in the future. For this reason, the waterfall model often induces a feeling of frustration in customers and developers when requirements are not well defined, not complete or simply not doable.

Other software development models tried to solve the waterfall model problems with to a varying degree of success. Most of them directly inherit the principles of waterfall model. Some examples are:

- **V-model**: A decomposition of the problem phase (verification) is followed by a testing phase (validation). The two paths, one going down and the other going up, shape a V as showed in Figure 7. Between those two, the technical activities are found [10].

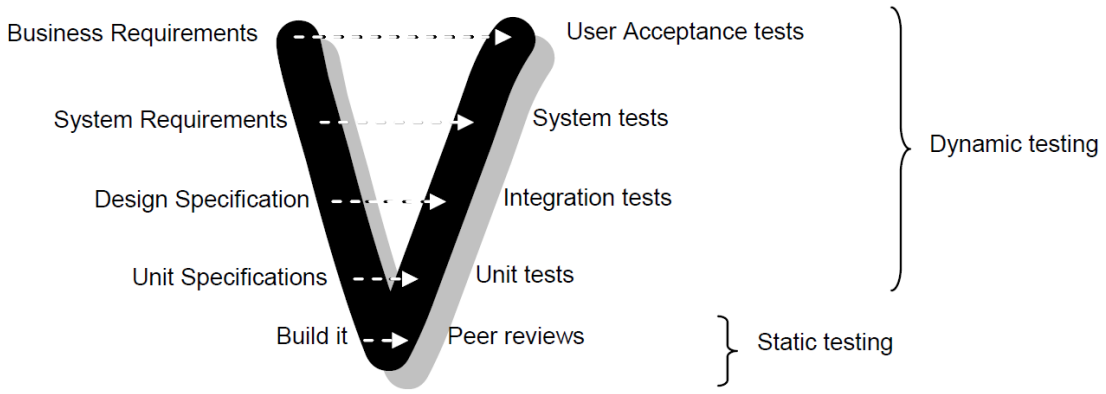


Figure 7: V-model
Source: Development with the V-Model [10]

- **Spiral model**: Originally conceived for projects driven by risk, this software development model includes a specific risk analysis phase. It also includes the development of software prototypes closely related with simulation and testing stages which were not present in the original waterfall model [11]. The model is graphically represented in Figure 8, where this special phase is found in the top right quadrant.

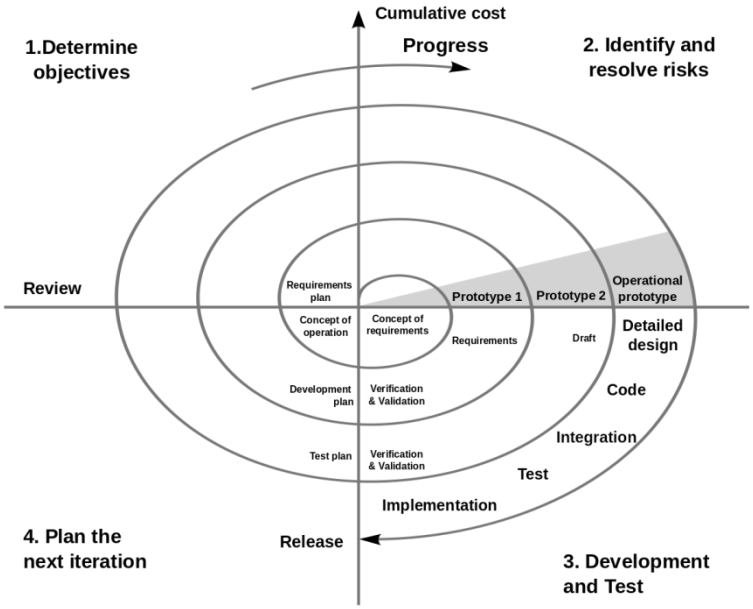


Figure 8: Spiral model
Source: Boehm [11]

- **Iterative and Incremental Development model**, **IID** (multi-waterfall cycle): Keeping the well-known waterfall model phases: analysis, design, implementation, testing and evaluation; the process is repeated several times and receives feedback from its previous iteration, building a more robust software product each time the cycle is completed. The user involvement in the development process will finally shape the agile software development model as explained in the next section.

3.1.2 Agile Model

The agile software development model is "*a reaction to traditional ways of developing software*" [12] that responds to the need of an alternative to documentation driven, heavyweight software development processes. The origins and principles are found in the Agile Manifesto:

> "We are uncovering better ways of developing software by doing it and helping others to do it. Through this work we have come to value:
>
> • Individuals and interaction over process and tools
> • Working software over comprehensive documentation

> • Customer collaboration over contract negotiation
> • Responding to change over following a plan
>
> That is, while there is a value in the items on the right, we value the items on the left more" [13].

Customer satisfaction is the first principle of agile model and it is achieved by delivering small pieces of valuable software in a short period of time (weeks). Therefore, the agile model is partially based in IID model with the characteristic iterative and incremental approach. This continuous releasing of working software is the measurement of progress in agile. The iteration objectives must be kept small enough to be released with such a cadence. Simplicity is essential.

The customer or stakeholder is involved in the development process, which means requirements are volatile and developers should be prepared for the changes. In fact, changes are welcome in agile and developer's initiative is valued. Initiative in the development team is boosted by the team lead, who is responsible for creating a work environment where technical excellence is reached. The communication with the customer and between developers is performed preferably face-to-face.

The advantages of agile software development model can be summarized as:

- **Early and continuous delivery**: the first functional software does not take long time to arrive to customer hands. Iterations are short and the risk of a complete failure is very low (assuming the previous deliveries went ok).

- **Customer satisfaction**: due to the inclusion of the user in the development process, he can decide what to do and when to do it. When the project is running, the user understands (helped by the developer) what type of functionality takes more time to be developed and therefore raises the costs. With those variables in hand, he can prioritize the work.

- **Change is allowed**: requirement changes may happen and the team is ready to process them in a fast way. Design changes in the code (refactor) are also welcome as they are considered very important for the software evolution.

- **Improved product quality**: motivation and implication of a technically strong development team improves the quality of the outcome software [14].

However, these benefits provided by the usage of agile methods cannot completely replace traditional methods in the coming years. Diversification in the software development processes is necessary [15]. On the one hand, aspects such us big teams management, developers leaving the project, or highly critical domains (human lives endangered) require strict quality control, so traditional methods will still apply. On the other hand, innovative projects with unclear requirements carried out by small teams fit perfectly with agile models [16].

Horus and the application developed in this project are both driven by an agile software development model. There are specific actions taking place inside the team that make it agile, such us daily standups, weekly releases, peer reviewing and team retrospectives. The following section will provide some more details on agile model.

3.1.3 Agile Methodologies

The specific Agile Methodologies used in this project are introduced below. In addition to a theoretical exposition, it is also explained how these agile methodologies are being followed inside the Horus team.

3.1.3.1 Kanban

Kanban is a methodology to visualize the work flow with the objective of limiting the work in progress. It has its origin in Toyota's assembly-line manufacturing, when David J. Anderson formalized the method to be applied in software development in 2010 [17]. Kanban is translated as "visual signal" in Japanese, and work items in this methodology are represented exactly like that, visual items, also known as "Kanban cards"; placed in a "Kanban board".

Kanban cards represent work as a visual item, allowing the developer and the team to track the work progress and flow. The card should contain enough information to be understandable by the rest of the team members, so everybody knows what the work of the others is. Description of the work, responsible and time estimations are good attributes for a well-defined Kanban card.

Kanban board contains several columns that represent queues of prioritized work. It is the single source of truth for the team and the representation of Kanban's transparency. The developer moves work items from one column to the other managing its own workflow. Sharing progress in the team is usually done via stand-ups supported by the Kanban board. This enables developers to explain what they are doing based on the work items present in the board.

A basic Kanban board should have at least a three-step workflow represented by three columns: *To Do*, *In Progress*, and *Done*. Structure of Kanban board may vary depending on the team's size, structure and objectives [18]. In Figure 9, an example of a Kanban board containing some work items can be seen. In the figure, there is also an example of a Kanban card, with some representative details.

Anderson identifies in his book the five key principles of Kanban:

1. Visualize and manage the workflow via Kanban board and cards.
2. Finish over start which means to limit work in progress (WIP).
3. Make continuous, incremental and evolutionary changes (Kaizen).
4. Everyone understands the process.
5. Collaborate in order to improve.

Anderson's approach is considered rather large, formal and bureaucratic, but it is the main reference for Kanban in the software development industry. Another approach, more suitable for this project, is given in *Personal Kanban* [19]. Benson and Barry believe Kanban can be applied in very small teams or even for personal usage. This method is interesting in our case since the author of this End of Master Thesis Project has done the work by himself.

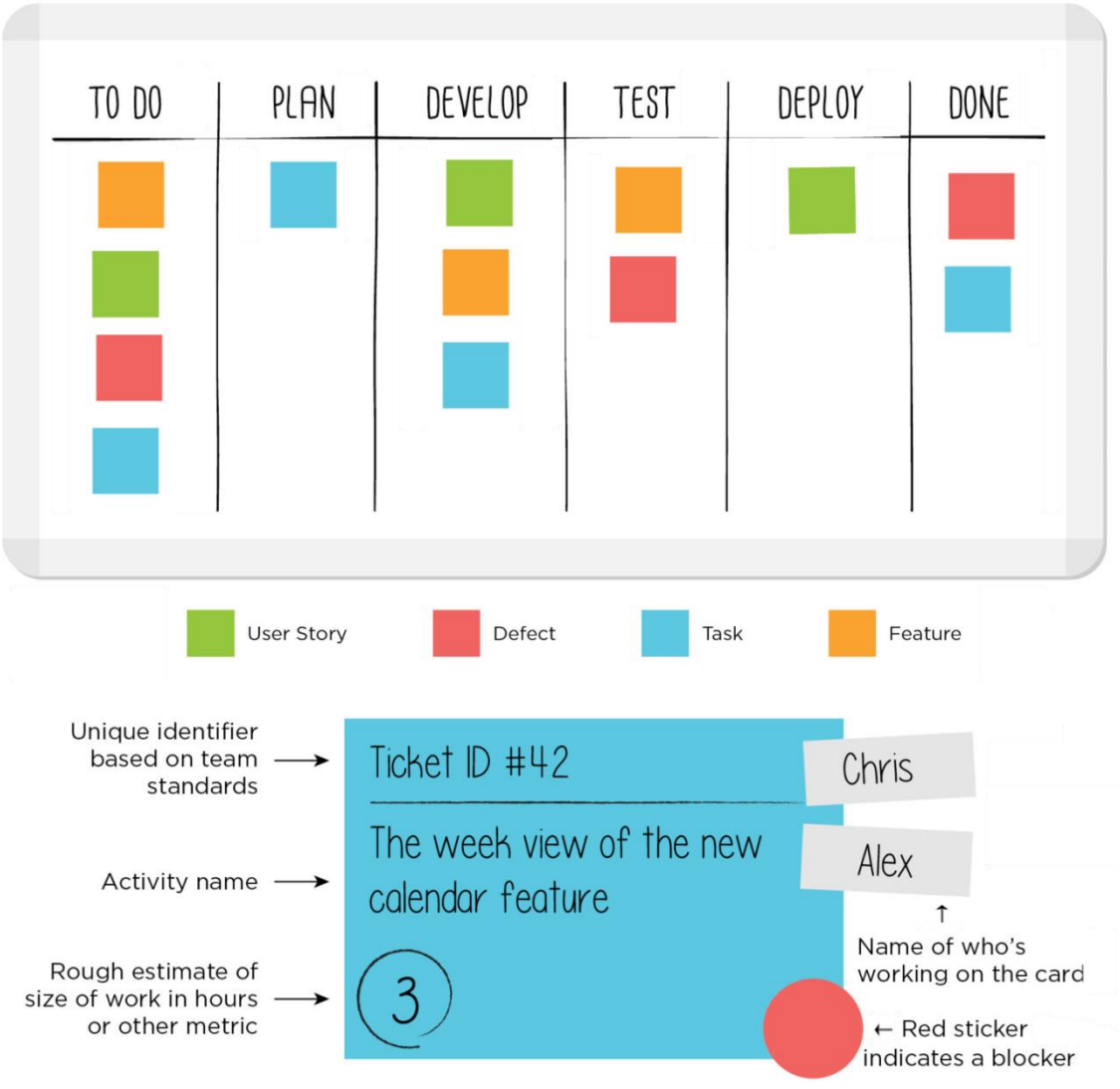


Figure 9: Example Kanban board and Kanban card.

Source: Leankit, Kanban [Modified] [20]

Kanban is a scheduling system, a way of organizing "Work Packages" and making workflow visible, but does not give any indication about how to do the work. Therefore, an agile software development methodology should be defined containing more specific guidance for technical development. FDD is the methodology used in Horus and by extension in this project.

3.1.3.2 Feature Driven Development

Feature Driven Development (FDD) is an agile methodology that defines an adaptive software development process driven by small pieces of fully functional software. Created by Jeff De Luca as project manager in Singapore in 1997, it defines a pragmatic approach to develop enterprise software. "*When done well, FDD produces meaningful, accurate and timely status reporting and progress tracking for all levels of project leadership*" [21]

FDD is described in five phases:

1. **Analyze problem domain**: the team should invest just enough time at the beginning of the project to build an object model having sufficient details to start producing work in the backlog stage.
2. **Produce features**: using knowledge from process 1, the team starts making some to-do features, small software pieces of functionality valuable for the customer. These features fill the backlog.
3. **Plan features**: the set of features defined in the previous stage, are now prioritized, estimated and assigned to a developer.
4. **Design feature**: assigned features are now designed in order to meet customer needs. This can be an individual or a team activity.
5. **Build feature:** the code activity finally starts. Testing tasks and code reviewing are also part of this last process [22].

The main advantage of this methodology is its short iterative process that produces tangible working results frequently. The customer is very pleased when he sees something working after the first week or two even if it is something rather simple. This is a situation observed quite often during the development of **12NC Timeline**

FDD methodology in the Horus team means:

- Meet the customer in order to understand the problem domain. There is no need to know all details related with the business, just enough to start working.
- Communication with the customer will produce features to fill our backlog.
- Weekly meetings with the team to plan new features. The knowledge of the team gives a broad perspective and a more accurate estimation. Use collaborative tools like Atlassian JIRA and Bitbucket to aid feature tracking, knowledge exchange and source code reviewing.
- Design is often made by the developer and the development is supported by:
    - "Git branches" for each feature being developed (see 3.2.1.6), so the current working version is always the starting point of the development.
    - "Feature databases", a quickly deployed copy of the production database to develop features that alter the database.
    - Continuous integration build server (Jenkins in this case) to facilitate the feature releasing and test automation.

The technologies mentioned above are explained in the next section.

3.2 Tools

In this section the tools used for the project development are introduced. They can be classified into two categories: development tools and scheduling tools.

3.2.1 Development tools

Development tools include programming languages, software development platforms and collaborative tools.

3.2.1.1 Scala

The main programming languages in Horus applications are PHP [23] and Scala [24]. PHP was used during the first years of development. Now the project is shifting to Scala because it offers several interesting advantages. This project was addressed as a new application, so most of it has been programmed using Scala.

Scala is a general purpose programming language that combines object-oriented and functional programming concepts with a strong static type system. There are four aspects that make Scala unique:

- **Compatibility**: Scala programs compile to JVM bytecodes which means they are fully compatible with Java code. In fact, Scala can call Java methods, inherit Java classes, etc. In that sense, a new developer can see Scala as a way to add value to his existing Java code.
- **Conciseness**: A program written in Scala is very likely to be shorter than the same program written in Java. Code 1 and Code 2 illustrate an example:

```java
// Java
boolean nameHasUpperCase = false;
for (int i = 0; i < name.length(); ++i) {
  if (Character.isUpperCase(name.charAt(i))) {
    nameHasUpperCase = true;
    break;
  }
}
```

Code 1: Example Java code

```scala
// Scala
val nameHasUpperCase = name.exists(_.isUpper)
```

Code 2: Example Scala code

The programs above are checking if `String` variable `name` contains an upper case character. It is easy to see the difference in the length of code written. This property is also known as "expressiveness" or the capacity of the language to express more using fewer words. In addition, fewer lines of code does not mean only less typing, it also means less code to review, less typos, etc. Programs become more tidy and easy to read.

- **High abstraction level**: Code 1 and Code 2 are also good examples of Scala abstraction capacity. Java treats strings as low-level entities composed by characters and uses a for-loop structure, while Scala treats strings as high-level sequences of characters and makes use of function literals (`_.isUpper`). Similar level of abstraction is also possible to achieve in Java, but it requires many more lines of code. [25]
- **Static types**: static type system is considered by supporters of dynamic languages (Ruby, Python, etc.) as a not nice restriction for expressing themselves when coding, but it has some important advantages. Static types prevent certain run-time errors like: integers being concatenated to a set of strings, booleans being added to integers, functions applied to an unfitting number of arguments, etc. In addition, static types allow a safer refactoring, for example when changing the number of arguments in a function; the compiler will show error in the appropriate places.

3.2.1.2 Play Framework

Play Framework [26] is an open source development framework for building high-productivity Java and Scala web applications. It makes use of the widely known architectural pattern Model-View-Controller (MVC). Play is stateless, there is no session per connection like in Java EE; and asynchronous, requests are served asynchronously. Following those reactive principles, Play is suitable for delivering responsiveness and resilience applications.

3.2.1.3 TreeGrid

Although many other components are included in Horus and in this software project, TreeGrid [27] represents the main core of 12NC Timeline application functionality so it is worth to be mentioned here.

TreeGrid is a JavaScript component that generates dynamic HTML code to display and edit data in a table, grid or tree view. The possibilities are almost endless with this package; it allows all kind of operations to organize the data. A convoluted example taken from the official TreeGrid website can be seen in Figure 10.

The appearance of TreeGrid resembles to MS Excel and users that are used to the MS tools find it very familiar. Not all functionality available in TreeGrid was used for this project but some of the main available features were implemented:

- Column resizing, hiding/displaying, spanning, etc.
- Tree feature in columns (expandable columns)
- Advanced filter system and row sorting
- Paging mechanism for displaying large amounts of rows
- Fixed (frozen) columns and rows
- Export formats XLSX, XLS, CSV or HTML tables

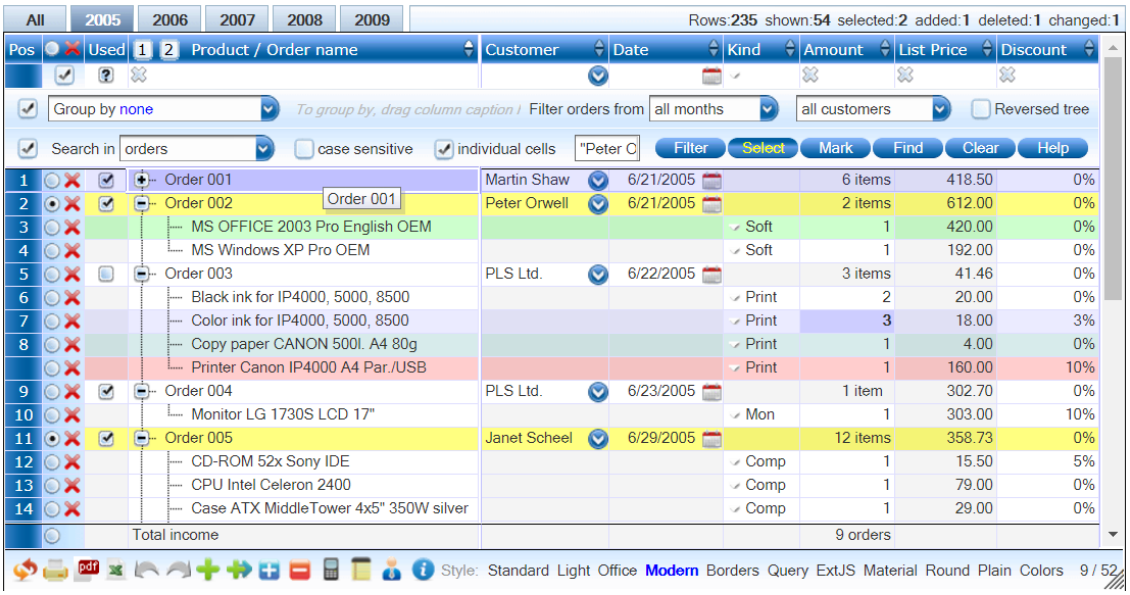


Figure 10: TreeGrid example
Source: Editable JavaScript TreeGrid [27]

3.2.1.4 IntelliJ IDEA

An integrated development environment called IntelliJ IDEA [28] from JetBrains was chosen for development activities. It includes a plugin for Scala language coding and Play Framework. This IDE supports a lot of nice features that makes developer's life easier, such us automatic code completion, reliable refactoring tools, built-in version control tools (Git), etc.

3.2.1.5 MySQL

As pointed out in 2.3, Horus applications are importing and exporting information continuously, and they need a proper storage to keep the more than 100GB of data they manage. MySQL [29] is the most popular open-source database management system. It is used in this case to organize the structured data collection in Horus.

MySQL databases are relational, that means they store the data in separate tables. The table structures are organized to optimized speed. Tables contain rows and columns that can be governed with a set of rules defined by the user. Rules can be defined to maintain data consistency avoiding, for example, duplicated or missing values.

"SQL" in MySQL name stands for "Structured Query Language". SQL is a standard language to access databases. It can be used: directly in a database browser, for example to retrieve data from the database and use it later for reporting purposes; embedded into source code, for retrieving data to process in the application; and behind an API that hides SQL syntax, for developers not experienced with the language. In this project, SQL was used mainly embedded in the source code allowing some data processing.

Approximately half of the development work spent in **12NC Timeline** application has been invested in activities related with MySQL and SQL (designing and coding queries). Therefore, it is considered as one of the most relevant tools used in this project.

3.2.1.6 Git

Git [30] is a free and open source Distributed Version Control System (DVCS). It is quite popular nowadays in the software development industry due to its fast performance compared with other similar tools. It is based in a distributed management system in which every developer has a local copy of the complete history of changes.

Git's workflow is divided into four different stages as shown in Figure 12. There is a remote repository with the metadata and the objects Data Base of the project. There is also a local repository containing a copy of the remote repository. The developer needs to update the local repository manually. From the local repository, the developer can generate a workspace to make changes in the files. The staging area is just an intermediate step and it contains the files ready for being copied to the local repository.

Workspaces in Git are defined as *branches*. A branch is an independent line of development. An example of a Git tree can be seen in Figure 11. In this tree, several branches with some *commits* (circles) can be found. A commit is a change confirmed in the repository and it is identified with an ID. With the tree structure, a team of developer can collaborate and work independently in different branches. Upon completion of their work, they can merge back the branches to the master branch that represents the main line of development, first locally and then pushing their changes to the remote repository.

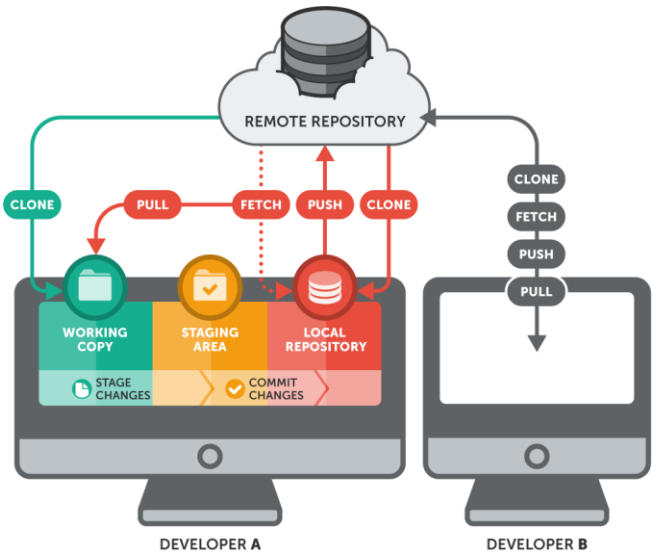


Figure 12: Git repositories

Source: Tower Git [37]

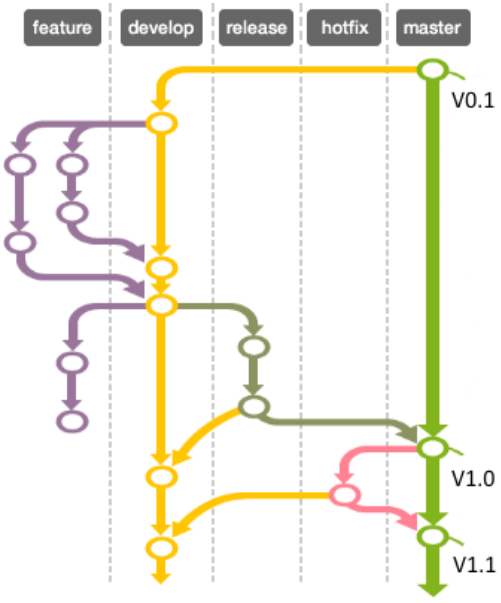


Figure 11: Git branches

Source: Backlogtool Git guide [37]

3.2.1.7 Bitbucket

The next tool introduced is Bitbucket [31], closely related with Git. It is a web-based application property of Atlassian and it is used mainly for code reviewing. Bitbucket is integrated with Git system, which allows the developer to create "pull requests" based on the branches in the remote repository. Then, these "pull requests" are reviewed by a team member. The "pull request" system is really useful as a collaborative tool because remarks on the code can be done in the web browser without pulling the branches from the repository. Bitbucket also includes a functionality to merge branches when the code review is finished.

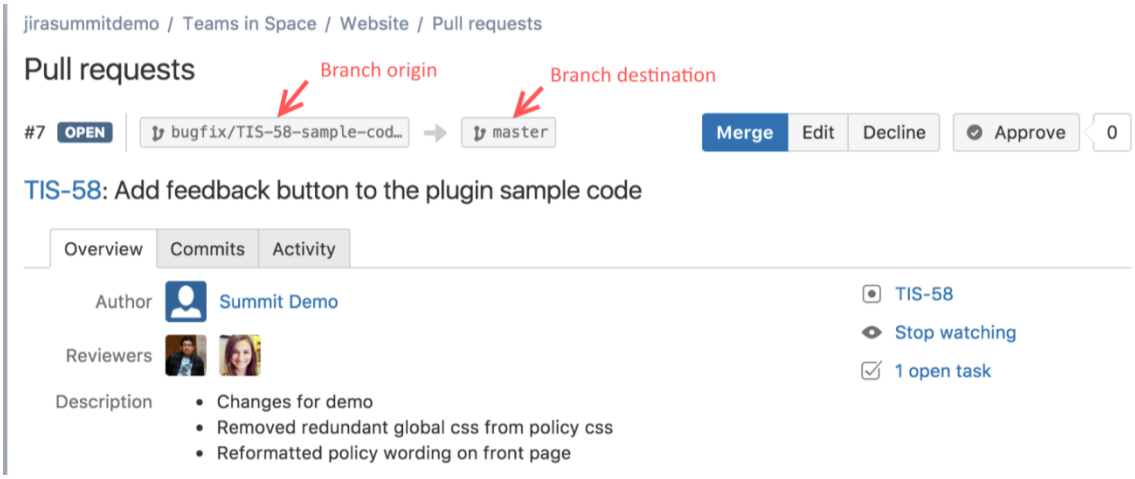


Figure 13: Pull request example

Source: Atlassian Bitbucket [31]

3.2.1.8 Jenkins

Continuous Integration (CI) is a software development practice based on the automatic daily compilations and test runs of a software project. It also streamlines processes as the generation of installation packages or mock testing websites. Jenkins [32] is an open source automation server which provides continuous integration capabilities in a software project. It can be easily integrated with version control tools such as Git. The compilation and test processes are in fact usually configured to be run when detecting changes in the repositories (*commits*).

Jenkins plays an important role in Horus because it is used to generate the *Production Site* and the *Feature Sites*. These concepts are closely related with the Kanban methodology. The *Production Site* is the definitive instance of the Horus project where the functionality is supposed to be perfect and in accordance with user requirements. The *Feature Sites* are instances of the Horus project running in a development environment. The developer can share with the final user a link to one of this *Feature sites*, containing a new functionality he had just developed. Thus, the customer can test it before it is finally integrated in the definitive instance.

3.2.2 Planning tools

The planning tools were used mainly to track Work In Process (WIP) and to communicate with the client.

3.2.2.1 JIRA

JIRA [33] is a web application for software project planning. It helps to track issues and to schedule actions inside the development team. It is an Atlassian tool like Bitbucket. Its board layout is perfect to see how the work elements are flowing through the development process. JIRA allows the user to define a lot of different work element types. These are the ones used in Horus:

- *Stories*: new features of the product which have yet to be developed.
- *Bugs*: problems which impair or prevent the functions of the product.
- *Epics:* aggregation of related stories.

The steps these issues are going through can be set by the tool administrator, but they are generally: *Backlog*, *In progress* and *Done*. Therefore, JIRA is also an excellent tool to implement some of the Kanban principles.

In Horus, where Kanban agile methodology applies, JIRA is used for tracking the WIP. Furthermore, Git branches representing user stories or issues are labelled with a JIRA ID in Horus team by convention. In this way, the project changes can be easily documented to share knowledge with other team members or clients. Appendix "A1. Project Plan" reflects these JIRA IDs and a general project plan. More details about JIRA and how it was used to make **12NC Timeline** project plan are given in the following section.

4 Work Plan

The definition of a detailed Work Plan for a project, like the one described in this document, is not an easy task. **12NC Timeline** application development started with a high-level set of requirements, and it took a certain time to clarify them. However, the definition of an adaptive Work Plan can add benefits, as it facilitates setting realistic goals. It can be considered as an agreement between the stakeholder and the developer. The details about the Work Plan definition are presented in this section.

4.1 Work Breakdown Structure

The first step when setting up a project should be defining a Work Breakdown Structure (WBS) definition. The objectives defined during the initial analysis are then translated into a concrete set of "Work Packages" with a schedule, a responsible, a reviewer, etc. The WBS definition in *agile* software projects can be certainly considered as volatile in most cases, but never as less important [34]. The way of setting realistic goals is through estimation. Estimation is a procedure extensively practiced inside *agile* teams because it helps to make agreements with the customer. It is often based on experience and expertise, so discussions with several members of the team are promoted because they are both beneficial and productive.

In this End of Master Thesis project, although the development work was carried out entirely by the student, suggestions coming from other Horus team members regarding estimations were introduced in order to organize the work in a better and more efficient manner. The estimation activity inside Horus team is made in a particularly interesting way that receives the name "*Planning Poker*". It is a consensus-based estimation technic used by *agile* teams [35]. In this game-like activity, each team member receives a set of cards with numbers on them, representing time units (usually days). One pending request is selected from the backlog and the responsible explains the issue to the others. Afterwards, everyone selects a card from his hand and shows it at the same time. The mean of all values is then selected as the time estimation for that story. Estimations made with "Planning Poker" result in very accurate timing. A "Planning Poker" hand is showed in Figure 14.

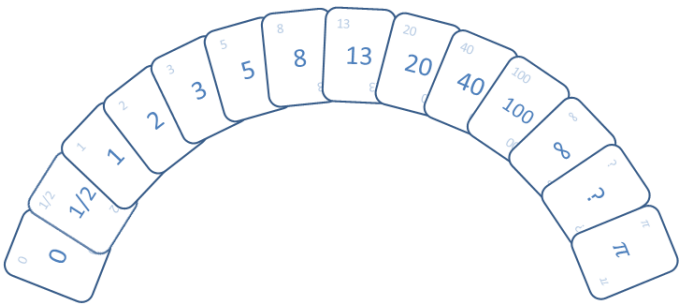


Figure 14: Planning Poker hand
Source: Planning Projects with Poker [35]

The basic WBS for **12NC Timeline** was defined using this technique. Its status at the starting point is shown in Figure 15. Several "Work Packages" were combined to build up *Epics* (see section 3.2.2.1) until the application was complete. The work is organized on a weekly basis, although the weeks are not in this case as the ones from the calendar, but packages of 25 working hours due to part-time schedule. The initial estimation of effort for the above mentioned WBS was approximately 300 working hours. In general terms, these initial assumptions were fulfilled as expected, which is reflected in appendix "A1. Project Plan".

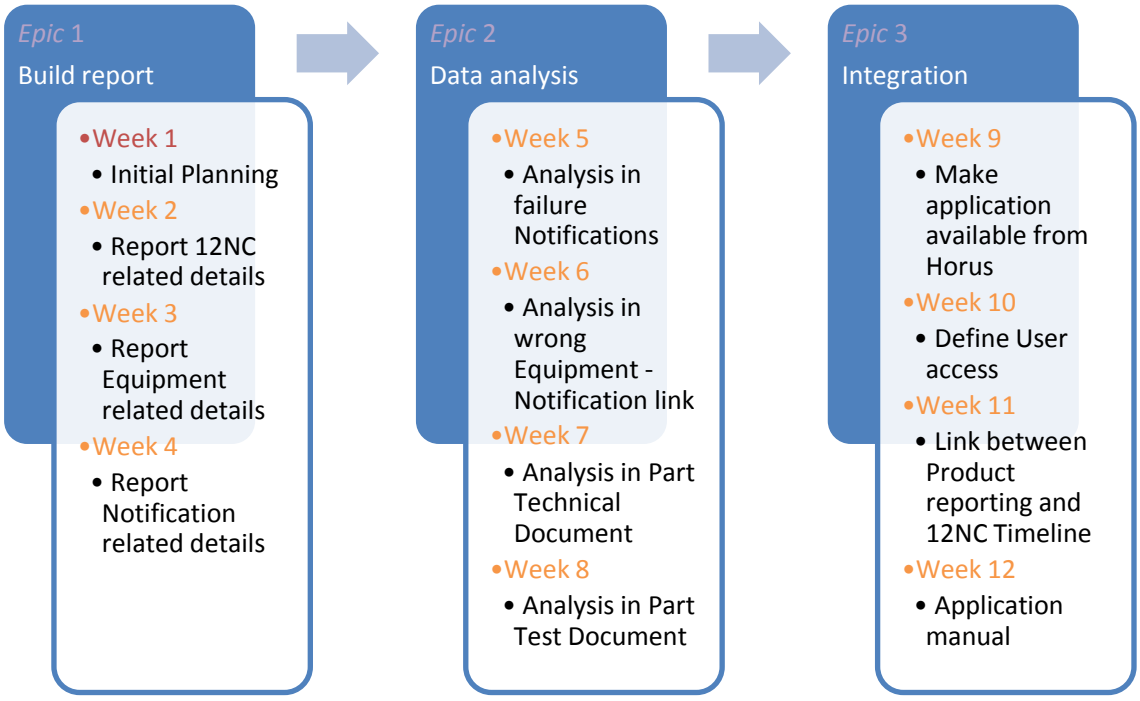


Figure 15: Initial planning and *Epics* content
Source: Own representation of initial planning

A weekly meeting with stakeholders was scheduled on top of these identified activities in order to ensure alignment with their requirements. The weekly "Work Packages" were later subdivided into smaller "User Stories" or "Features". These elements were delivered in accordance with an interactive FDD process. More details about the actual organization of the work are provided in the next section.

4.2 Work plan in JIRA

The tool to create detailed WBS for this EMT Project and to manage its development has been JIRA, introduced in section 3.2.2. JIRA allows a wide range of configurations in order to fit different types of agile methodologies. The one selected in Horus team is Kanban, as explained in section 3.1.3.1. The way of working using JIRA inside Horus team is basically as follows:

1. The administrator creates a Kanban board where issues are displayed in columns indicating their status. This configures the project workflow which is entirely customizable by the administrator. An example of JIRA Kanban board for **12NC Timeline** can be seen in Figure 16. The meaning of the columns is as follows:

    - **Backlog**: issues not to be addressed immediately (less than a week). It can contain an undetermined number of issues.
    - **Selected for Development**: issues to be addressed immediately (this week). It can contain a relatively large number of issues (up to 10).
    - **Waiting**: issues that cannot be resolved due to lack of critical information. It is a step back from the "In Progress" stage.
    - **In Progress**: issues under development. The number of issues in this column is limited to 3. It helps limiting the WIP.
    - **Reviewing/Testing**: issues under review. In order to carry out the review activity, these issues are expected to have an open "Pull Request" linked to them (see section 3.2.1.7)
    - **In Integration**: issues that passed satisfactorily the review stage. Issues "In Integration" will be merged into the production environment in the next release. There is no way back from this column.

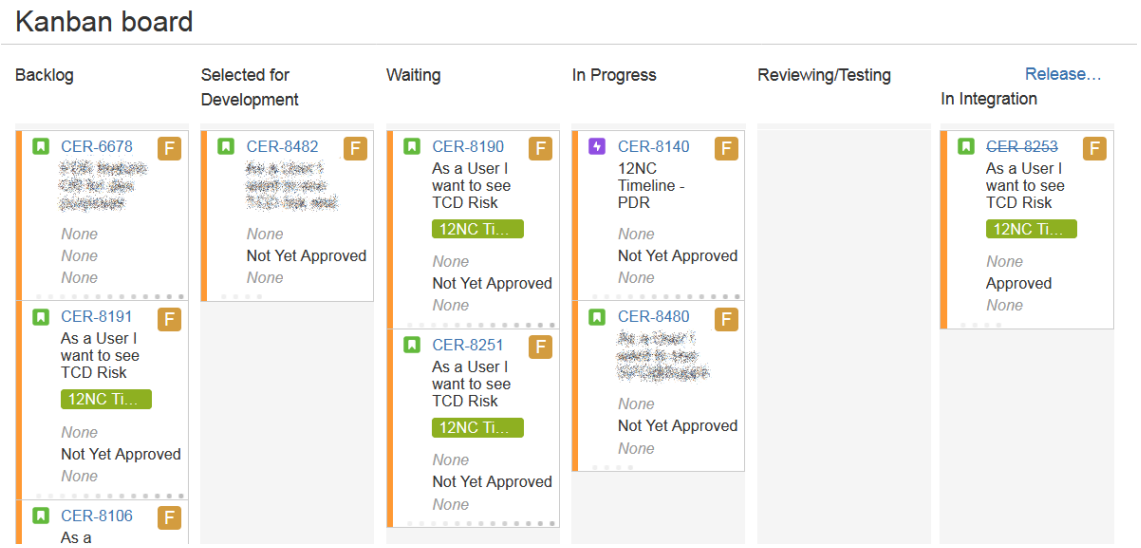


Figure 16: 12NC Timeline JIRA Kanban board example
Source: Screenshot from JIRA application

2. The developer can create new issues or manage the ones already present in the board. The developer cannot make changes to the workflow stages.

3. In order to create an issue, it is necessary to provide a summary, short description, responsible stakeholder, responsible developer and time estimations. There are three issue types: *Stories*, *Bugs* and *Epics*, as described in section 3.2.2.1.

4. The new issues are automatically added to "Backlog" column just after being created. The development does not start until they reach the "In Progress" status.

5. When issues reach state "In Progress", they are dispatched following FDD process (see section 3.1.3.2). This process is showed in Figure 17 for this particular project and it represents the core activity in the development of an issue. It has a cycle structure because it is repeated for every feature until an *Epic* is released. The explanations of the steps are as follows:

    - **Initial planning**: It is the very first step, when developer and user sit together in order to discuss the general objectives of the *Epic*. This includes timings and general schedule.
    - **Feature requirements**: Developer and user meet to define requirements about a specific feature, usually a story that will not last more than a week.
    - **Feature design**: The developer translates the requirements into application functionalities. An experienced team might help in this step.
    - **Feature implementation and testing**: The developer implements the changes in the application. Afterwards, he (or preferably another team member) tests these changes via manual or automated tests.
    - **Feature reviewing**: The developer asks a team member to review his work.
    - **User approval**: The developer asks the user to check the changes implemented in a test environment. If the user agrees, changes are merged into the production instance of the application. Finally, the process starts again with a new feature until the *Epic* is completed.

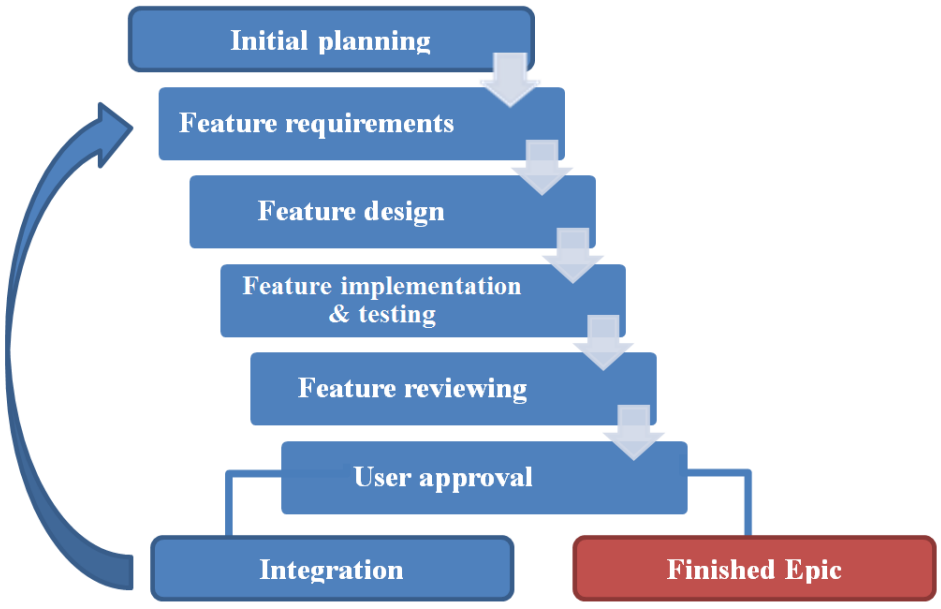


Figure 17: FDD process applied

Source: Own representation of Feature dispatching process

6. Different types of reports and summaries can be generated from the information about the issues in the board, e.g. average time an issue stays in waiting phase, number of issues completed per release, etc.

5 Development

The evolution of **12NC Timeline** development process is shown hereinafter. As described in Work Plan (see section 4) the development process was driven by three main milestones. The deliverables coming from each and every of them were functional fragments of the whole software package, suitable for immediate use. This is a simple and direct consequence of the flexibility coming from the *agile* methodology used in this case (described in section 3.1), a significant advantage for both the developer and the final user. Therefore, this section will be divided according to these three *Epic* (aggregation of related stories) releases.

NOTE: Because of confidentiality issues, some images presented in this section may have some blurred areas. This keeps information from data source safe, but does not affect the purpose or understanding of the images.

5.1 First *Epic* release

Like in any other IT project, the user knows at the beginning what he wants, but not in a clear and detailed manner, and definitely he does not know exactly how to do it. The developer needs first to understand the intentions and then translate them into functional requirements to check their feasibility. Afterwards, he has to assess the effort (time) to invest in their design and schedule it. These tasks are collective by nature and even experienced teams need a significant amount of time just to clarify the landscape. Even the name of the project is sometimes changed along this initial period, as it was in this case: at the beginning, everybody agreed to use 12NC as name root, and 12NC-CV ("CV" coming from *Curriculum Vitae*) was used. In fact the 12NC-CV name can be seen several times in the first conversations maintained with the user, but **12NC Timeline** was finally created two months later.

As a starting point, several meetings were needed in order to acquire the business notions related with the problem to solve. The tool to be developed was defined in these first meetings as an analysis tool to have a quick insight on 12NC. The processes taking place inside the organization are quite complex and, although the developer does not have to know everything in detail, at least some general knowledge and global concepts understanding is needed. Part of this general review of business knowledge is now the content of sections 1 and 2 in this document. Furthermore, in these first meetings the way to display the information was selected. Several options were considered as stated below:

- **Plain line**: items are displayed chronologically one after the other. It is the easiest way, but without the chance to apply filtering parameters it might be impossible to manage some parts with a large number of different items (attributes, shipments, etc.)

- **Table**: similar to the previous one, but with the most relevant items identified and organized in columns. This allows filtering and sorting mechanisms.

- **Graphical line**: items are displayed in different shapes surrounding a central line. It is a common way for displaying historical information that can be seen in text books. Filters would be available globally. It is the fanciest way among the options discussed but it requires the usage of graphical libraries and more discussion about formatting details.

Figure 18 shows the board with details about the displaying possibilities after one of the first meetings. White boards were regularly used in the development of the project and they proved to be an efficient tool for clarifying ideas and centralizing discussions

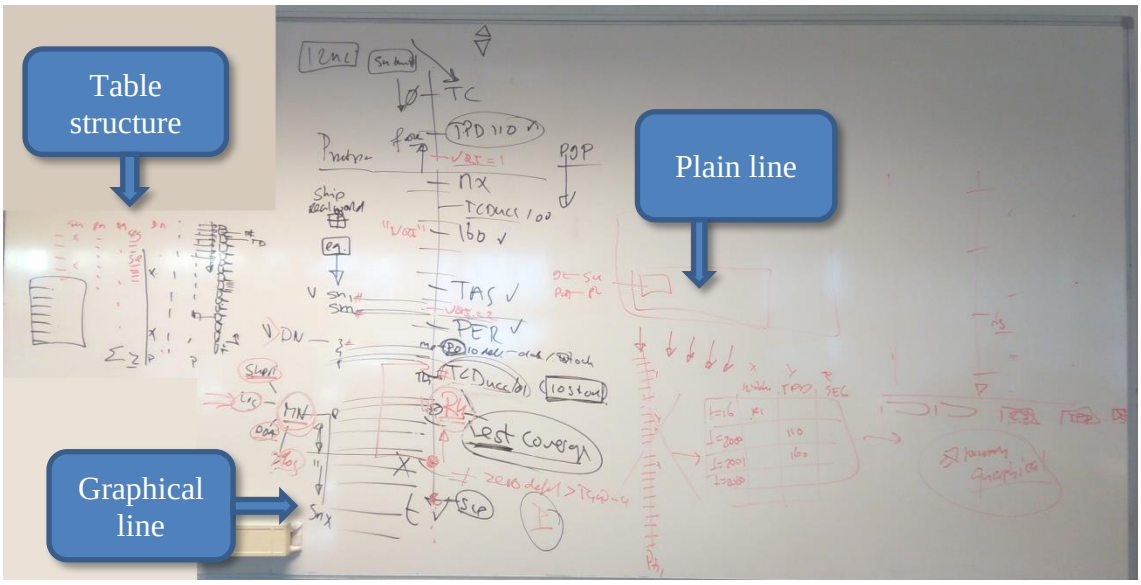


Figure 18: Discussion board in the first meetings
Source: Picture taken during first meeting

Finally, a table structure was chosen. It has the advantage of being familiar for the users as they worked daily with Excel worksheets. This structure is also easy to implement using TreeGrid (see section 3.2.1.3), a technology available in Horus. The TreeGrid library is used in several applications inside Horus and provides a solid base for report and application development.

During the initial phase of development, special attention was paid to find the data to be managed. The finding of data proved to be of increased difficult due to the variety of information storage systems used in the sponsor company (see section 1). As mentioned in section 2.3, SAP and TC are the most important data source systems utilized by Horus. The set of part basic attributes to be reported was also defined during the first weeks. This led to the agreement on general 12NC attributes, such as date of creation, associated design and production documents, importance indicator, etc. as well as more specific information like *SAP Equipments* and *Notifications*. These two last terms will appear several times along this section. The definition of this set of attributes allowed delivering some functionality only two weeks after the development had started. Details about the results are given in section 5.1.2.

A *SAP Equipment* is a physical individual occurrence of a generic part. It is identified with an Equipment Number (EN) and the 12NC of the part. It might be a product already delivered, still under manufacturing or under testing process before delivery; or a component received from a supplier. In all the cases, it is an item which may originate issues to track. The information and the number of *SAP Equipments* related with a part might be extensive, and sometimes not easy to manage. The difficulties found while working with *SAP Equipments* data can be found in the 5.2.1 section.

A *Notification* is a record describing an issue related with a part. It is usually written by an engineer on information generated inside the company from manufacturing, testing, and other internal organizations; or outside from a supplier or a customer. Hence, data contained in a *Notification* is very important in order to improve internal processes. Knowing what went wrong is an effective way to learn, but this type of information is also sensitive because nobody wants to publicly share his failures. Therefore, the access to it is restricted to authorized users. One of the key points of *Notifications* is to support issues traceability. To preserve basic data consistency, the 12NC indicated in a *Notification* should be the same as the one indicated in the linked *SAP Equipment*. Unfortunately, this is not always the case as explained in section 5.2.1.

There is a report already present in the current version of Horus to show all the details about a part: the "Part Info Page" (PIP). It is one of the most important pages in the Horus suite due to the large number of daily users. It contains information that covers a wide range of part attributes: from the details about the documents related with it, to the list of the most important suppliers. In some cases, the information displayed shows dates of events related with the part, but some historical attributes are missing. It also includes information about the *SAP Equipments* and *Notifications* of the part as shown in Figure 19, but in an unlinked way, showing only the connections between 12NC – *SAP Equipment* and 12NC – *Notification*. The connection between *Notification* and *SAP Equipment* is not shown, so there is no traceability between them unless some time-consuming digging is done on their content by user. That is one of the reasons why **12NC Timeline** plays now an important role complementing the information contained in the PIP.

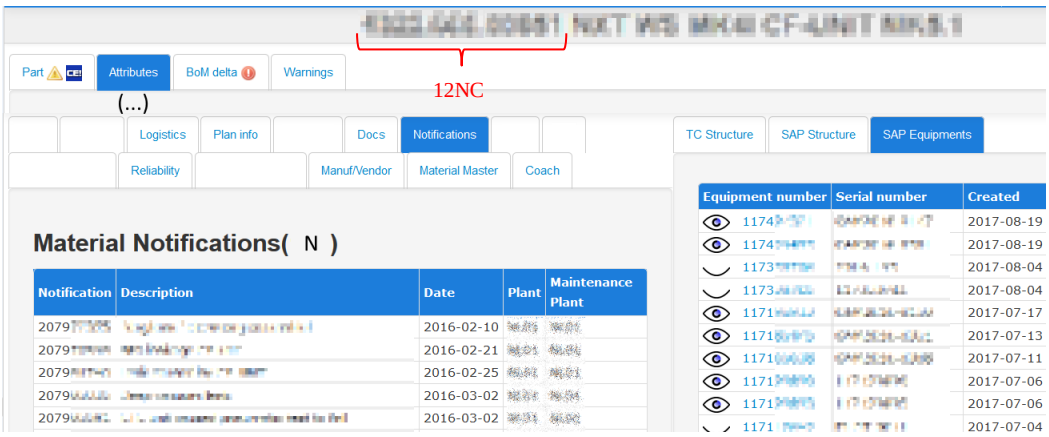


Figure 19: Part Info Page | Notification & SAP Equipment tab

Source: Screenshot taken from PIP application

5.1.1 Challenges 1

After defining the displaying method (table) and some data to be reported (12NC general details, *SAP Equipments* and *Notifications*), requirements were clear enough to start the development. The main challenge encountered during the first *Epic* development was to find the information to be reported. The majority of the data is easily found in Horus databases because they were already imported from the data sources (mainly SAP). Nevertheless, not all the information required about *SAP Equipments* and *Notifications* was there. Therefore, some import mechanism had to be developed. Thankfully, due to good coding practices and project organization, the development of new importers is relatively easy in Horus.

Import tasks are run daily by Horus servers in order to keep warehouse information up to date. The process followed by the importers is called "Extract, Transform and Load" (ETL) and it is a very common process in business intelligence systems. A graphical representation of this process is shown in Figure 20. SAP was selected as the data source in this example but importers do not work inclusively with SAP data. These are the steps executed by the import tasks:

1. **Extract**: Source system (i.e. SAP) is queried and the requested information is stored in an intermediate Tab-Separated Values file (TSV). Then, a "raw" table is created in a volatile Horus database and it is filled with the data coming from the TSV file. The number of columns and their order in the file is decided upfront to match the "raw" table structure.

2. **Transform**: Data in the "raw" table is processed, for example by modifying the data types of some columns (string to date) or applying some prepared functions (split data in a column). Once done, transformed data is stored in a table.

3. **Load**: Data is loaded from the "transformed" table in the final Horus table. The table to be filled in Horus database is dropped and created again or it can be updated with the appropriate statement.

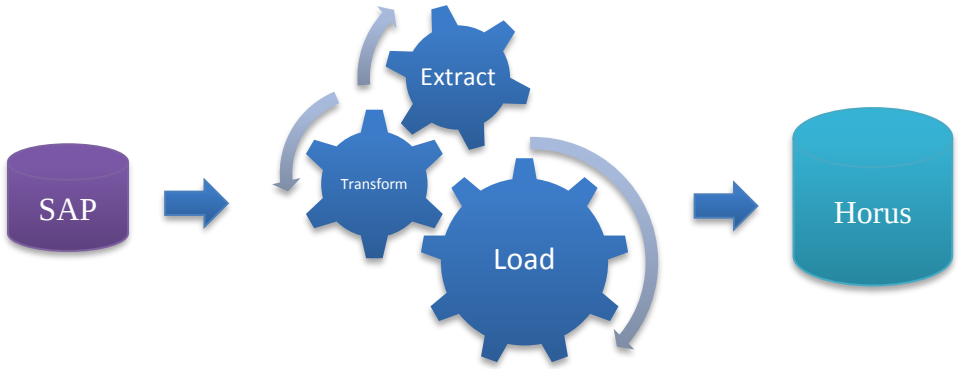


Figure 20: ETL process

Source: Own representation of ETL process

5.1.2 Results 1

The first version of **12NC Timeline** was released by February 9th 2017. It was only possible to access this version after knowing the page URL, and this was only shared with some key-users. The application was not available in Horus menu until *Epic* 3. Figure 21 shows the Version 1 of the application. The user can select the 12NC in the selection box located at the top-left of the page. The items displayed as a table are *SAP Equipments* linked with *Notifications*, with the last as the primary element. The relation between them is the EN.

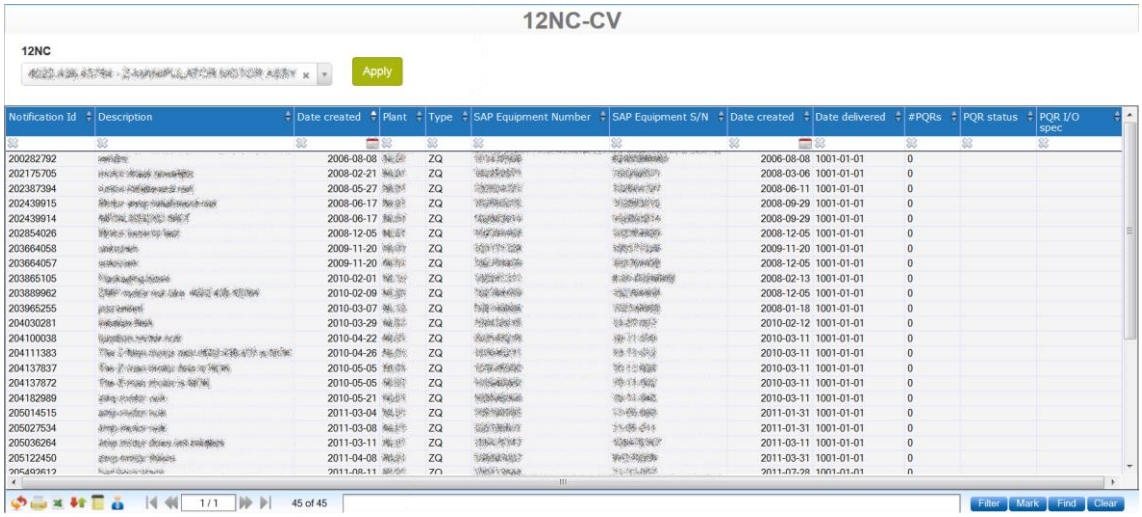


Figure 21: 12NC Timeline V1
Source: Screenshot taken from 12NC Timeline application

The approach in Version 1 could be improved because there are some *SAP Equipments* with no *Notifications* attached, so they simply do not appear. Even with no issues to solve, it is useful to display them just to check data consistency. That means, a better way of reporting this information would be putting *SAP Equipments* as the primary element and reporting *Notifications* linked with them. Apart from that, it was decided to add some more information. Figure 22 shows the structure of the report once changed.

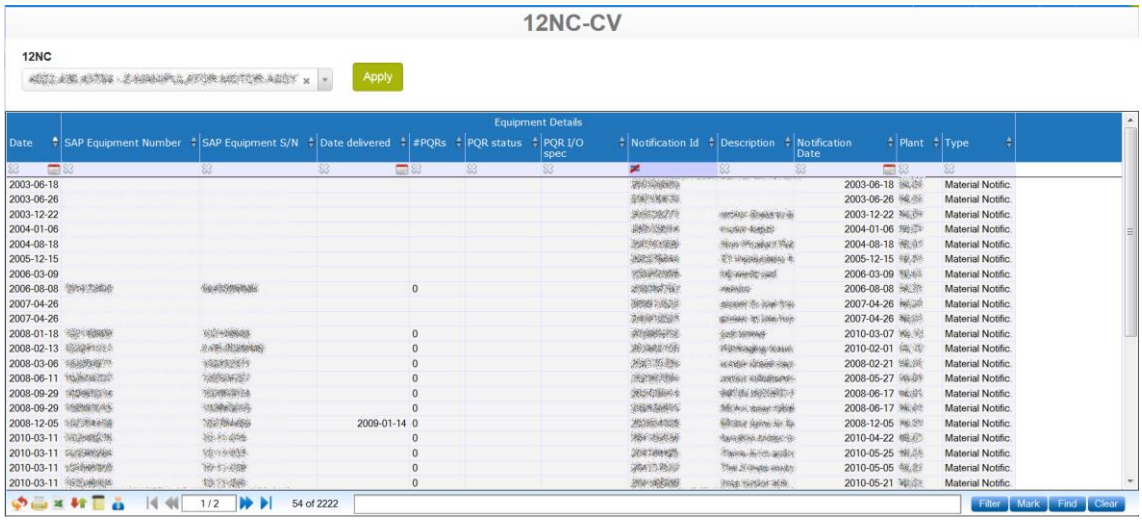


Figure 22: 12NC Timeline V2
Source: Screenshot taken from 12NC Timeline application

To finalize this first *Epic*, a Version 3 was developed adding 12NC Details section. These details are attributes related with the part, such as the creation date (12NC Creation), the design development phase where the part is located, the volume qualification indicator (VQI) and the documents related with the part (Doc Infotype). These attributes can be found in the PIP, but the user has to click in several tabs to get the data. Having them in a single view in **12NC Timeline**, eases understanding and achieves time savings. *Notification* section now has its own header, too. Figure 23 shows the result after these changes had been applied.

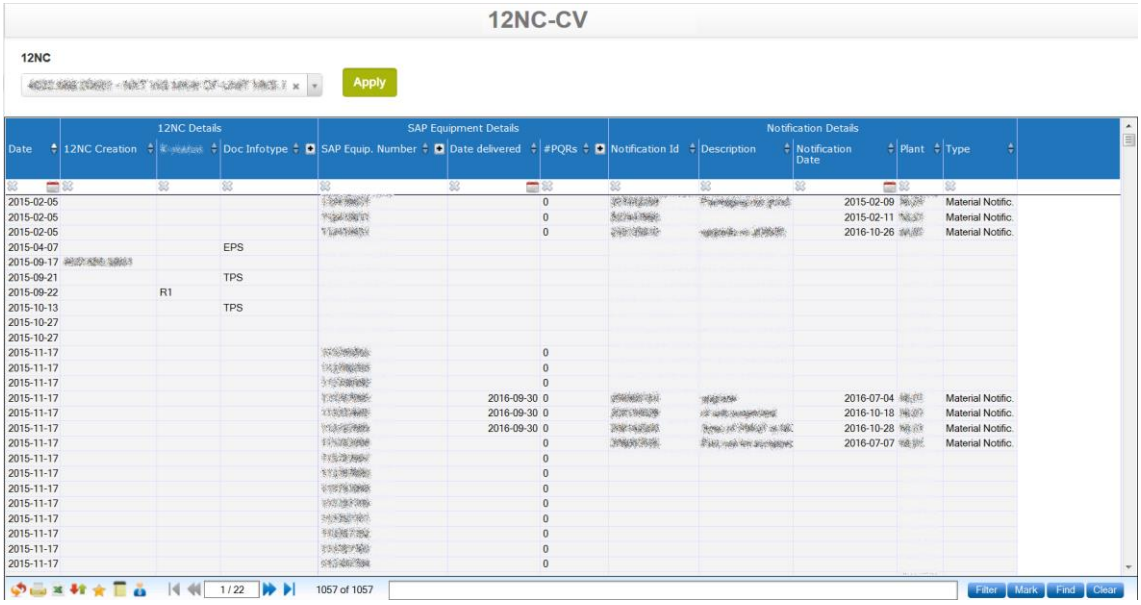


Figure 23: 12NC Timeline V3
Source: Screenshot taken from 12NC Timeline application

The differences between Version 1 and 3 are remarkable. First of all, a date column is placed in the left side of the report. That column represents the essence of the timeline. All the attributes in the right side of the report have a date in the left side. The "Date" has a different meaning depending on the attribute. For *SAP Equipments*, it means the creation date of the item, and for VQI it means the modification date of the attribute. In addition, a two-level header is included. It allows a better organization of the data reported by grouping related attributes together.

Another important improvement is the inclusion of expandable/collapsible columns. Using this TreeGrid feature, it is possible to group several columns into one and to make the report more compact. If the user is particularly interested in an attribute within an expandable column, he can click on the plus icon next to the column name (⊞) and then look for it. Figure 24 shows the behavior of these expandable columns.

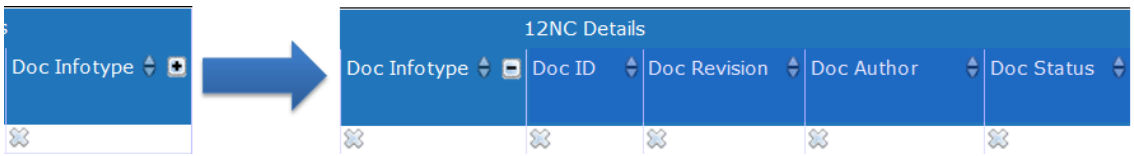


Figure 24: Collapsible columns functionality
Source: Screenshot taken from 12NC Timeline application

5.2 Second *Epic* release

The objective of this second phase was to perform some analysis on the data reported. Nevertheless, this analysis was combined with the addition of some more attributes. Thus, the report continues its expansion in this phase.

The analysis on the data reported has the objective of mitigating the risks along the Part Lifecycle. One way of reducing the risks is by paying attention to the information contained in *Notifications*. As said in section 2.2, a MN provides information about issues appearing once the part was shipped and was being used by the customer. If issues are found during the utilization of a part, it means the testing process, through which the part went after production, was not sufficient to ensure it will work properly. The document that covers this testing process is called TCD. All data analysis in **12NC Timeline** revolves around the TCD content.

The TCD completion is the last step before the part goes to production phase because this document covers theoretically every possible aspect needed to be tested by the supplier before delivering a part to the sponsor company. If there are some MNs after the TCD was marked as done, there is a problem, because it means the TCD was not complete enough. On the other hand, there are parts that do not have a TCD because they are not considered critical (VQI setting equal 0 or 1), but if there is a MN for any of them, it means a TCD is needed to prevent the error from occurring again. There are several documents that are necessary in order to write a correct TCD. Two of these documents are TPD110 and TPD160. If, by any chance, these documents are not present when the TCD is finished, it means there is something wrong. A TCD cannot be complete without the feedback of these TPDs, but sometimes engineers miss this.

The situations mentioned above suggest the creation of tracking mechanisms to control them. The name these trackers received in this project is “Warnings”. These “Warnings” are the result of the Part Lifecycle data analysis and will help the user to detect variations in the normal flow of the part industrialization process.

5.2.1 Challenges 2

One of the main problems to solve in this second phase was the lack of data consistency in some of the storage systems. In this case, inconsistency affects the information about *SAP Equipments* and *Notifications*, and the connection between them. The topic of this section is how this problem was managed.

A *Notification* indicates an issue affecting an individual *SAP Equipment*. Therefore, among the data contained in a *Notification*, the individual identification is needed just to ensure traceability. This identification is the EN. Both *Notifications* and *SAP Equipments* hold a connection with a part since they contain a 12NC among their properties. This is somehow redundant because the *Notification* could be connected with the part through the *SAP Equipment* 12NC in a transitive relationship.

This triangular relation is shown in Figure 25. All these data are loaded manually in SAP and unfortunately this is not an error free process. A simple way of checking data consistency is to compare the number of *Notifications* in the PIP and in 12NC Timeline, or equally the number of *SAP Equipments* (see Figure 19). If the numbers from both sources for any of them are not equal, there is something wrong.

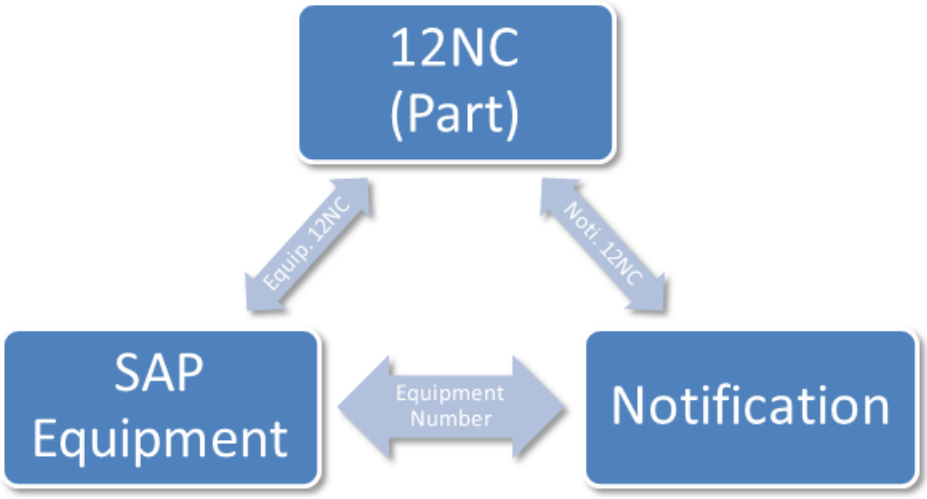


Figure 25: SAP Equipment - Notification | Link diagram

Source: Own representation of data link

After some investigation, it was discovered the links between the three elements were not always right, meaning the data sources had been manually filled with incorrect data.

Several situations may appear when using **12NC Timeline** to get the information on a given 12NC. A summary of all these situations is offered to the reader in appendix A2. Data consistency. They are also explained below:

1. There are *SAP Equipments* associated to the part with no attached *Notifications*. This is the standard situation when *SAP Equipments* do not have any *Notifications*. There are no errors, data are consistent and there is no problem to be solved.

2. There are *SAP Equipments* associated to the part with attached *Notifications* containing a valid EN and the right 12NC. This is the standard situation when *SAP Equipments* have some *Notifications*. When an individual *SAP Equipment* has several *Notifications*, there will be as many rows of information in **12NC Timeline** as number of *Notifications*. Data are consistent in this case.

3. There are *SAP Equipments* associated to the part with attached *Notifications* containing a valid EN and a wrong 12NC.

   In this situation, the number of *Notifications* in **12NC Timeline** is higher than in PIP because only *Notifications* for a specific 12NC are counted here, but there is no mean to find the *Notifications* containing the wrong data.

In order to mitigate this problem, an additional column with the 12NC contained in the *Notification* was added. In this way, a direct checking of "Equipment 12NC" vs. "Notification 12NC" can be done, and *Notifications* containing the wrong data can be found. Figure 26 shows this work around:

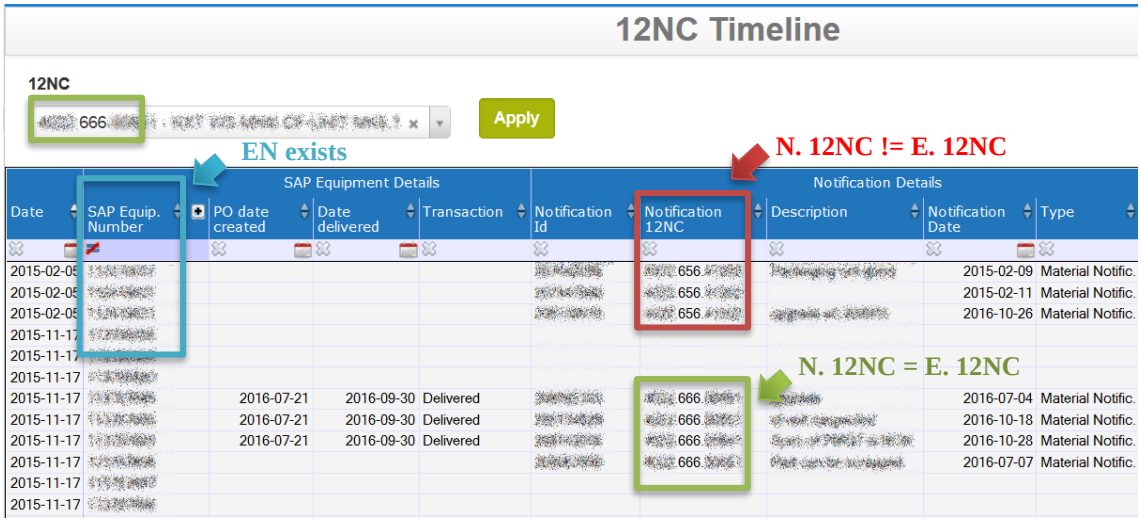


Figure 26: Notification 12NC column explanation
Source: Screenshot taken from 12NC Timeline application

4. Apparently, there are no *SAP Equipments* connected to the part because *SAP Equipment* Details columns appear empty, but there are *Notifications* attached whose "Notification 12NC" is the given 12NC. This "Orphan Notifications" appear when the EN was wrongly introduced or left empty by the user.

   In this situation, the number of *Notifications* in **12NC Timeline** is lower than in PIP because they are displayed here based on the 12NC, not taking into account their connection with a *SAP Equipment*.

   These "Orphan Notifications" are easily seen in **12NC Timeline**. Figure 27 shows an example with some of them.

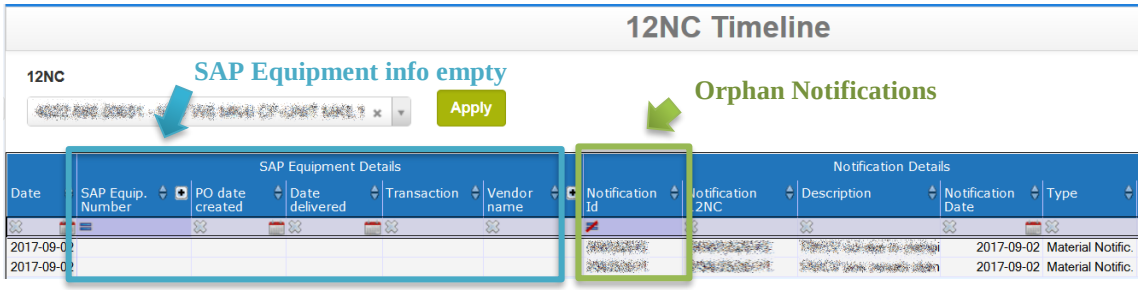


Figure 27: Orphan *Notifications*
Source: Screenshot taken from 12NC Timeline application

5. There are really no *SAP Equipments* associated to the part, nevertheless there are some *Notifications* whose "Notification 12NC" is the given 12NC. It is a case similar to point 4, but the error now is in the "*Notification 12NC*" instead of in the "EN". The result is equally an "Orphan Notification".

In this situation, as in point 4, the number of *Notifications* in **12NC Timeline** is lower than in PIP because they are displayed here based on the 12NC, not taking into account their connection with a *SAP Equipment*.

These "Orphan Notifications" are displayed in **12NC Timeline** showing no apparent difference to the previous point. Figure 27 represents also a valid example of a report with some of them. The information is obtained with a different query than the one used to retrieve the *SAP Equipments*.

The difference between both situations cannot be seen in the report generated by **12NC Timeline**, but as a result of error correction. In that described in point 4 there are real issues to solve, therefore the *Notifications* shows them and they remain in the report once the errors have been corrected, while in the situation described here, there are no issues to solve and the *Notifications* disappear as soon as the errors are corrected, because they should not have appeared.

6. There are no *SAP Equipments* associated to the part and no "Orphan Notifications" appear. This is a standard situation. The expected result is an empty report with no *SAP Equipment* or *Notifications* information.

5.2.2 Results 2

After the addition of several attributes, **12NC Timeline** reached its final form. In Version 4, it had 44 columns in total. Figure 28 shows the menu that allows displaying or hiding the columns. Due to the report width, it was necessary to use the collapsible columns feature from TreeGrid in most of them. The three sections already present in Version 3 (12NC Details, *SAP Equipment* details and *Notification* Details) grew and a new section was added to display the "Warning" messages.

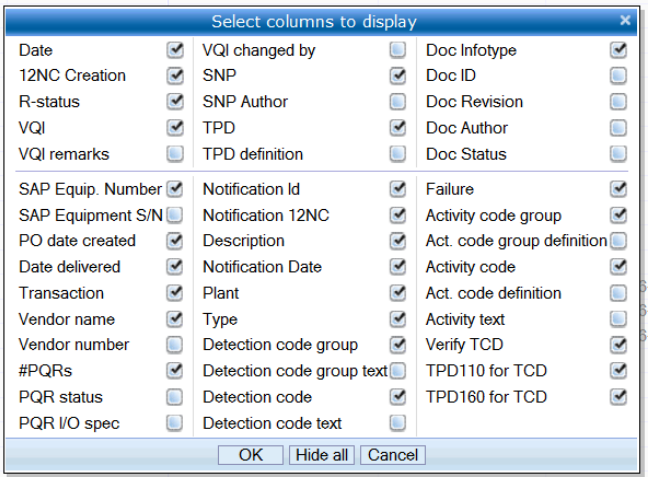


Figure 28: 12NC Timeline - Column display menu
Source: Screenshot taken from 12NC Timeline application

The columns added to 12NC Details section were Serial Profile Number (SNP) and TPD. The latter is used in the calculation of some "Warning" mechanisms. Figure 29 shows the final aspect of this section.

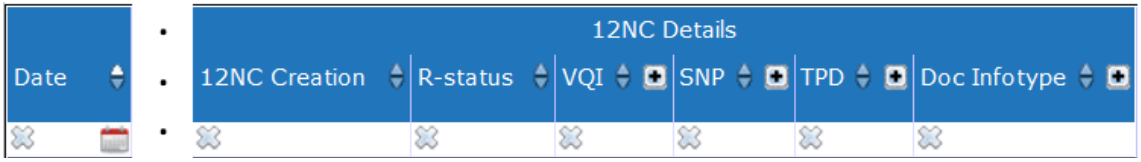


Figure 29: 12NC Timeline V4 - 12NC Details
Source: Screenshot taken from 12NC Timeline application

The columns added to *SAP Equipment* Details section were "Purchase Order (PO) date created" and "Transaction". The first one indicates when the *SAP Equipment* was ordered and the second one indicates if the transaction was a delivery or a return. Figure 30 shows the final aspect of this section.

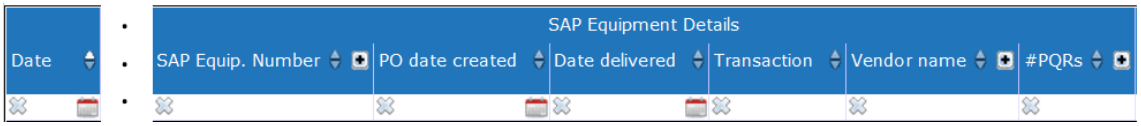


Figure 30: 12NC Timeline V4 - SAP Equipment Details
Source: Screenshot taken from 12NC Timeline application

The columns added to *Notification* Details section were "Notification 12NC", "Detection code", "Failure" indicator and "Activity code". The first one is used to show data inconsistency in *Notifications* data (see section 5.2.1). The other three are used in the calculation of "Warning" messages. These calculations will be explained later on. Figure 31 shows the final aspect of this section.

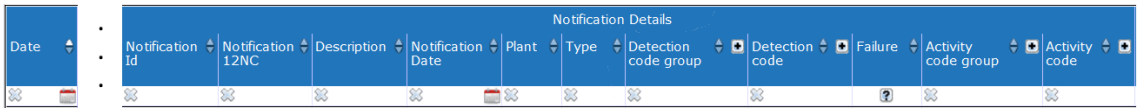


Figure 31: 12NC Timeline V4 - Notification Details
Source: Screenshot taken from 12NC Timeline application

The new section added in **12NC Timeline** Version 4 is "Warnings". In this section, three columns were added "Verify TCD", "TPD110 for TCD" and "TPD160 for TCD" as shown in Figure 32. These are the only columns in the report that are calculated and not directly retrieved from the database. The calculations were defined together with the stakeholders and represent the data analysis carried out in this application.

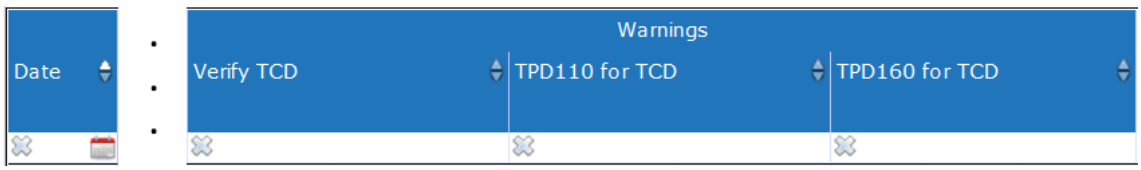


Figure 32: 12NC Timeline V4 – Warnings
Source: Screenshot taken from 12NC Timeline application

Calculations for each of the columns are as follows:

- **Verify TCD**: This column shows a warning message either when a "Failure" Material Notification is found in a *SAP Equipment* after the TCD was set to UCC status, or in a part with no TCD. This event means either the test coverage was not complete enough and the document requires some attention, or the document is missing but needed. The definition of "Failure" *Notification* is found in Horus database and is related with "Detection code group" and "Detection code" attributes. PO date also plays a role in the calculation, because if the *SAP Equipment* which has the "Failure" Material Notification was ordered before the TCD was set to UCC, it cannot be counted as a TCD problem because the document was not available when the order was made. Nevertheless, it is a situation to take into account. Table 2 shows examples with the possible Verify TCD messages:

12NC Details	SAP Equip. Details	Notification Details			Warnings
TCD UCC Date	PO Date	Date	Type	Failure	Verify TCD
-	01/05/2016	01/06/2016	MN	X	Failure MN and no TCD
01/01/2016	01/05/2016	01/06/2016	MN	X	Failure MN after TCD UCC
01/01/2016	01/01/2015	01/06/2016	MN	X	Failure MN after TCD UCC (PO older than TCD)

Table 2: Verify TCD warning messages

A prefix is added to the Verify TCD message in case the "Activity code group" and the "Activity code" shows that the *Notification* is either affecting a *SAP Equipment* coming from a supplier and under the responsibility of the Supplier Chain Engineer (SCE) department (prefix = *Supplier: …*), or that a SAP Equipment managed internally by the sponsor company (prefix = *Internal: …*).

Figure 33 shows an example of Verify TCD Warning messages in **12NC Timeline** application.

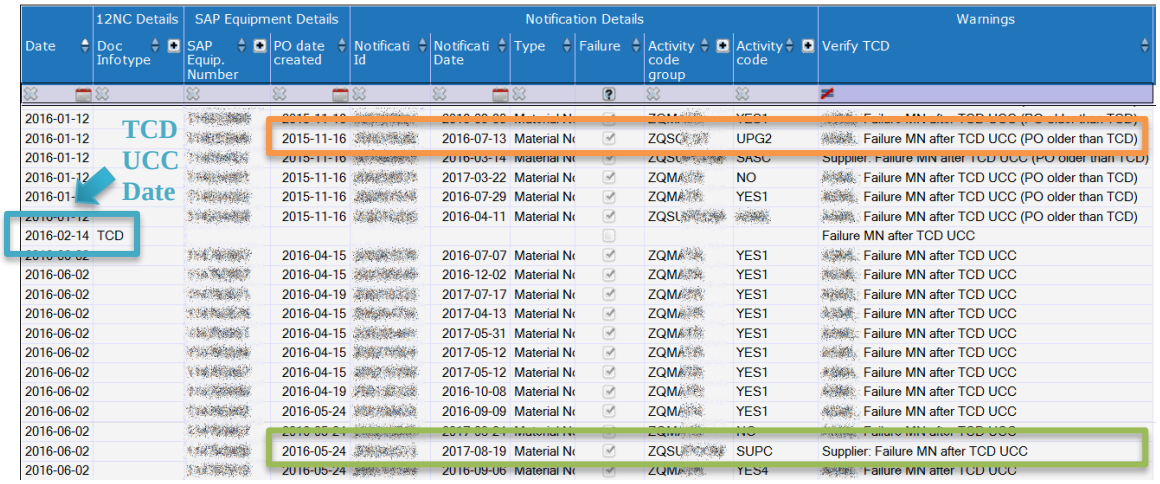


Figure 33: 12NC Timeline - Verify TCD Warning
Source: Screenshot taken from 12NC Timeline application

- **TPD110 for TCD**: This column shows a warning message about the status of the TPD110 when the part TCD is set to UCC. TPD110 is necessary feedback for the TCD and it should be in place. The "Warning" will reflect the document availability. Figure 34 shows an example of these warning messages.

	12NC Details		Warnings
Date	TPD	Doc Infotype	TPD110 for TCD
2014-05-02	110		TPD110 available from 2014-05-02
2014-06-24	110		TPD110 available from 2014-06-24
2014-06-24	110		TPD110 available from 2014-06-24
2015-11-10		TCD	TPD110 available since 2014-06-24

Figure 34: 12NC Timeline - TPD110 for TCD Warning
Source: Screenshot taken from 12NC Timeline application

- **TPD160 for TCD**: This column shows a warning message about the status of the TPD160 when the part TCD is set to UCC. TPD160 is a necessary feedback for the TCD and it should be in place. The "Warning" will reflect the document availability. Figure 35 shows an example of this Warning messages.

	12NC Details		Warnings
Date	TPD	Doc Infotype	TPD160 for TCD
2014-06-24	160		TPD160 available from 2014-06-24
2015-05-13	160		TPD160 available from 2015-05-13
2015-11-10		TCD	TPD160 available since 2015-05-13

Figure 35: 12NC Timeline - TPD160 for TCD Warning
Source: Screenshot taken from 12NC Timeline application

5.3 Third *Epic* release

After releasing Version 4, some actions were taken to promote the usage of **12NC Timeline** in the organization. The "Warning" mechanisms turned out to be really useful when analyzing Part Lifecycle problems, especially those related with the TCD. This has triggered exporting this data analysis information to other Horus reports.

As explained in section 1, the company manages assemblies composed by aggregation of parts. These items are called "Products" and there are some reports in Horus to display information about them. In most occasions, working with product information means simply managing the sum of information from parts below. For example, a Product Manager is interested in knowing how many parts inside the product have reached a certain design maturity level or how many documents related with the part are available. To cover this requirement, a Horus report was developed to offer a view where the parts inside the product are grouped according to the various possible design maturity states. This is the Product Deliverable Report (PDR).

From a Product Manager's point of view, **12NC Timeline** Version 3 (after second *Epic*) is not a very useful application because it is only showing details about a part. It is a too specific view. He is interested in a broader picture, asking himself questions like: What is happening in the thousands of parts that make up the product under my responsibility? Why is the final product causing problems when in use, despite all TCDs being in place? To answer these questions, it is necessary to bring the data analysis from **12NC Timeline** to the next level.

5.3.1 Challenges 3

The main challenge encountered during the development of this third *Epic* was not related to business knowledge, like in the previous *Epics*. Instead, trouble arose when changes in the technology used were realized, particularly in the programming language. PHP was used in this case of Scala, because the changes had to be implemented in some relatively old Horus reports, where this programming language was used originally approximately four years ago.

5.3.2 Results 3

Special attention was given by the sponsor company engineers to the TCD and the "Verify TCD" Warning from **12NC Timeline**. Therefore, it was decided to export the information from that column to other reports under the name "TCD Risk". Regarding a Product, this column reflects the number of parts that carry at least one Verify TCD warning message. With regard to a 12NC, it reveals the number of Verify TCD warning messages. The first case is shown in Figure 36 and the second one in Figure 37.

For products with thousands of parts, the calculation of the number of "TCD Risks" on loading time reduces significantly the performance of the existing reports. For that reason, it was decided to create an updater mechanism which would calculate those numbers for all parts on a daily basis, and that would store them in a table. This work is realized at night by a task running in Horus servers to avoid interfering with the users.

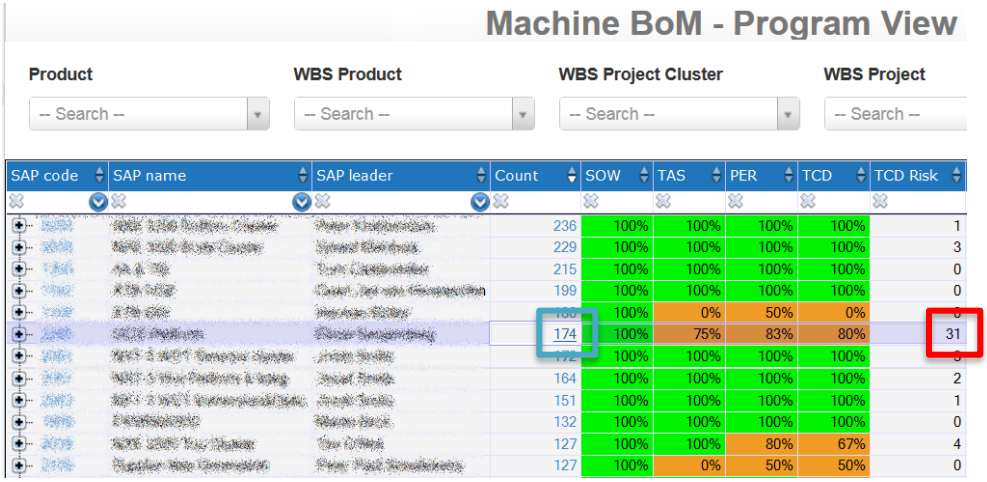


Figure 36: TCD Risk column in PDR
Source: Screenshot taken from Horus report

When opening the PDR report, shown in Figure 36, the user finds the "TCD Risk" column available and when clicking in the number displayed in the "Count" column, the "Details of Parts" page, showed in Figure 37, is displayed.

In "Details of Parts" page, lots of part attributes are available in addition to "TCD Risk". If the user wants more information about the number of "Verify TCD" Warnings, he can click on the link (yellow box in Figure 37) to access **12NC Timeline**.

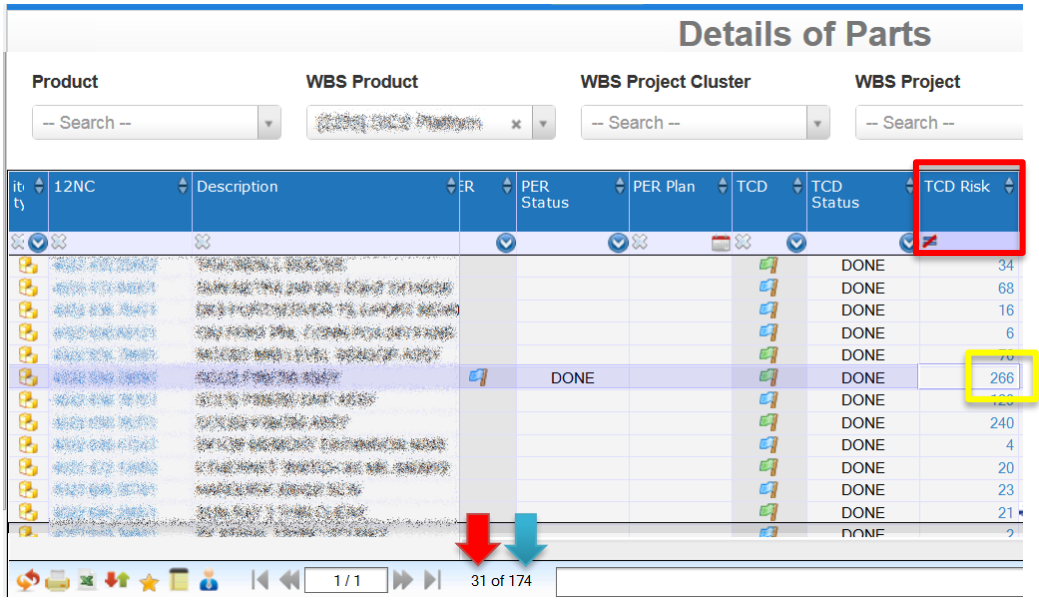


Figure 37: TCD Risk column in PDR Details of Part
Source: Screenshot taken from Horus report

When **12NC Timeline** is accessed via the last TCD Risk link, a special filter is applied in the report to show only the rows with the "Verify TCD" Warning set.

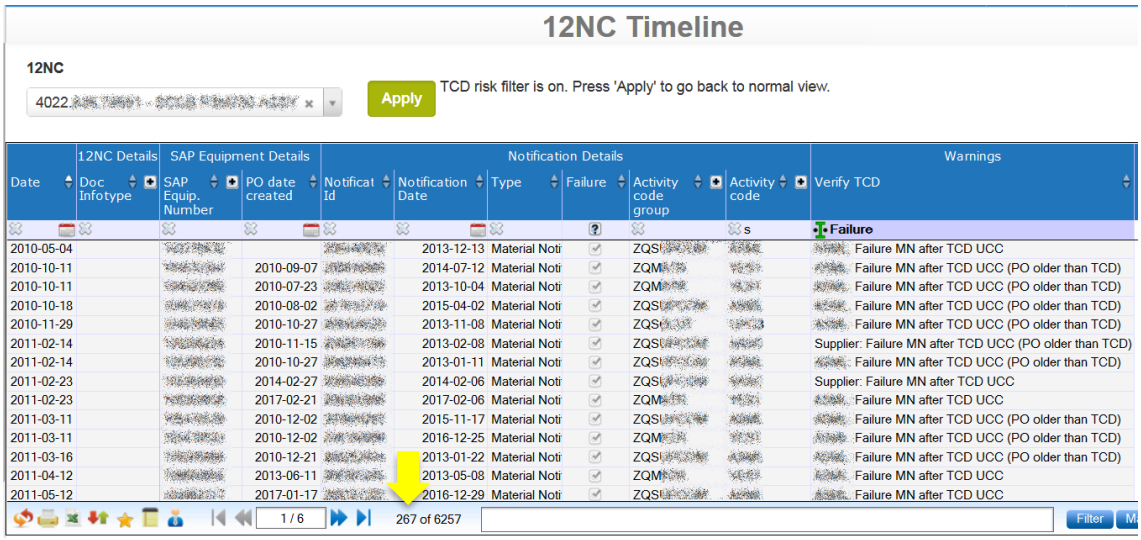


Figure 38: TCD Risk in 12NC Timeline
Source: Screenshot taken from 12NC Timeline application

6 Conclusions

Analyzing a product lifecycle can help to understand the given situation of a company and, thus, ease decision-making. Judgments are based on the analysis of data about a part's life. In modern companies, these are usually huge amounts of data, which make their analysis require the use of powerful IT tools. This project is a good example of how software technology can be used to solve a practical problem related with Product Lifecycle Analysis, helping a real company to improve product performance and internal processes through the analysis of information already available.

I. Achievement of the main objectives

As mentioned in section 1 the main goal of this project was to create an application that shows the entire lifecycle of a part inside a manufacturer of chip-making equipment. It has been demonstrated along previous sections of the document that this goal has been accomplished with the design and implementation of the application **12NC Timeline**, now in use as part of Horus suite. The project objectives, described in section 1.2, have been fully achieved allowing a much easier Part Lifecycle Analysis. The application is now working to the benefit of a real company. For that reason, the project can be considered successful from both the academic and the professional points of view.

II. Other Achievements

Developing a software application is connected to several difficulties. Sooner or later every development team faces the challenge to fulfill the user requirements on time, but those are usually not very clear and evolve continuously, while the time is always scarce. Agile methodologies appeared to help both user and developers in the construction of valuable software through a less restrictive method than in the classic ones (i.e. waterfall and others). By using these agile methodologies, it has been shown in this project how a sustained interaction with the user positively affects the development process. In addition, a Work Breakdown Structure with small "Work Packages" allowing greater flexibility has been proven to be an effective way of planning and implementing the work in this single-developer project.

III. Outlook

It is expected that user feedback will follow in the next months, allowing the application to continuously grow and expand its functionality for some time. The most probable improvements will be the addition of new part attributes in the report and possibly some new "Warning" mechanisms. The report structure improvement is another topic to be considered. At the project start some discussions took place about the style to display the information of the 12NC. Several proposals were considered and finally a table-like structure was chosen (see section 5.1). Nevertheless, this decision was taken because it was the easiest and fastest way to deliver some important functionality under the existing time and budget restrictions. In the future, this structure may be changed to get a more user-friendly display, like a graphic line of events.

7 References

[1] C. A. Mack, "Lithoguru," 2006. [Online]. Available: http://www.lithoguru.com/scientist/lithobasics.html. [Accessed 17 April 2017].

[2] A. J. den Boef, "Optical wafer metrology sensors for process-robust CD and overlay control in semiconductor device manufacturing," *Surface Topography: Metrology and Properties,* vol. 4, no. 2, 17 February 2016.

[3] T. Levitt, "Exploit the Product Life Cycle," November 1965. [Online]. Available: https://hbr.org/1965/11/exploit-the-product-life-cycle. [Accessed 10 May 2017].

[4] A. B. Khedher, S. Henry and A. Bouras, "Industrialization and manufacturing steps within the Global Product Lifecycle context," in *IFIP International Conference on Advances in Production Management Systems (APMS'09),* Bordeaux, 2009.

[5] J. Stark, Product Lifecycle Management, vol. 1, Springer, Cham, 2015.

[6] I. Sommerville, Software engineering, 9 ed., Pearson Education, Inc., publishing as Addison-Wesley, 2011.

[7] VersionOne, "Insights from 11th State of Agile Report," 11 May 2017. [Online]. Available: https://explore.versionone.com/state-of-agile/insights-from-the-11th-state-of-agile-report-2. [Accessed 15 July 2017].

[8] W. W. Royce, "Managing the development of large software systems," Proceedings of IEEE WESCON, 1970.

[9] Oxagile, "Waterfall Software Development Model," 26 July 2016. [Online]. Available: https://www.oxagile.com/company/blog/the-waterfall-model/. [Accessed 21 June 2017].

[10] A. Ratcliffe, "SAS Software Development with the V-Model," SAS Global Forum, United Kingdom, 2011.

[11] B. W. Boehm, "A Spiral Model of Software Development and Enhancement," TRW Defense Systems Group, 1988.

[12] D. Cohen, M. Lindvall and P. Costa, "DACS State-of-the-Art/Practice Report | Agile Software Development," Fraunhofer Center, Maryland, 2012.

[13] K. Beck, M. Beedle, A. v. Bennekum and A. Cockburn, "Agile Manifesto," 2001. [Online]. Available: http://agilemanifesto.org/. [Accessed 21 June 2017].

[14] A. Cockburn, in *Agile Software Development: The Cooperative Game*, 2 ed., Pearson Education, Inc., 2001.

[15] M. Lindvall and I. Rus, "Process Diversity in Software Development," IEEE, Maryland, 2000.

[16] A. Cockburn, "Selecting a Project's Methodology," IEEE Software, 2000.

[17] D. J. Anderson, Kanban: Successful Evolutionary Change for Your Technology Business, Blue Hole Press, 2010.

[18] D. Radigan, "How the kanban methodology applies to software development," Atlassian, 2016. [Online]. Available: https://www.atlassian.com/agile/kanban. [Accessed 05 February 2017].

[19] J. Benson and T. Barry, Personal Kanban: Mapping Work | Navigating Life, C. I. P. Platform, Ed., Modus Cooperandi, 2011.

[20] Leankit, "What is Kanban board?," 2017. [Online]. Available: https://leankit.com/learn/kanban/kanban-board/. [Accessed 26 May 2017].

[21] S. R. Palmer and J. M. Felsing, A Practical Guide to Feature-Driven Development, Prentice Hall, 2002.

[22] S. R. Palmer, "Feature-Driven Development (FDD)," 2014. [Online]. Available: http://www.step-10.com/SoftwareProcess/FeatureDrivenDevelopment/. [Accessed 11 May 2017].

[23] PHP Group, 2017. [Online]. Available: http://php.net/manual/en/intro-whatis.php. [Accessed 12 July 2017].

[24] M. Odersky, "Scala," École Polytechnique Fédérale, 2017. [Online]. Available: https://www.scala-lang.org/. [Accessed 14 June 2017].

[25] M. Odersky, L. Spoon and B. Venners, Programming in Scala, 2 ed., Walnut Creek, California: Artima Press, 2008.

[26] Play Team, "Play Framework," Lightbend, 2017. [Online]. Available: https://www.playframework.com/. [Accessed 17 June 2017].

[27] COQsoft, "TreeGrid," 2017. [Online]. Available: http://www.treegrid.com/Grid. [Accessed 19 June 2017].

[28] JetBrains, "IntelliJ IDEA," 2017. [Online]. Available: https://www.jetbrains.com/idea/. [Accessed 12 June 2017].

[29] Oracle Corporation, "MySQL," 2017. [Online]. Available: https://www.mysql.com/. [Accessed 14 July 2017].

[30] Open Source, "Git," 2017. [Online]. Available: https://git-scm.com/. [Accessed 14 June 2017].

[31] Atlassian, "Bitbucket," 2017. [Online]. Available: https://bitbucket.org/. [Accessed 13 June 2017].

[32] Open Source, "Jenkins," 2017. [Online]. Available: https://jenkins.io/. [Accessed 25 June 2017].

[33] Atlassian, "JIRA," 2017. [Online]. Available: https://www.atlassian.com/software/jira. [Accessed 18 June 2017].

[34] D. Rawsthorne, "Managing the Work in an Agile Project," Net Objectives, 2004.

[35] LeanMath, "Agile: Planning Projects with Poker and Pi," 10 April 2014. [Online]. Available: http://www.leanmath.com/blog-entry/agile-planning-projects-poker-and-pi. [Accessed 18 June 2017].

[36] Backlog, "Branching workflow using topic and integration branch," Nulab, 2017. [Online]. Available: https://backlogtool.com/git-tutorial/en/stepup/stepup1_5.html. [Accessed 20 June 2017].

[37] Tower, "Learn Version Control with Git," fournova Software GmbH, 2017. [Online]. Available: https://www.git-tower.com/learn/git/ebook/en/command-line/branching-merging/branching-workflows#start. [Accessed 10 July 2017].

Appendix

Additional information, hard to include in the main body of the document due to its extension, is offered in this section.

A1. Project Plan

The Project plan defined week by week is shown in the table below. The "Key" in the 1st column is the Work Package identification.

Horus Project Plan *12NC-Timeline*					
Key	**Type**	**Summary**	**Status**	**Estimated effort (h)**	**Due Date**
7735	Epic	12NC-CV TreeGrid	Released		**2017-03-20**
7653	Story	As a User I want to see actual deliverables linked with notifications in 12NC-CV	Released	16	2017-02-08
7736	Story	As a User I want to see details of events related with a SAP Equipment chronologically reported in 12NC-CV	Released	16	2017-02-15
	Task	As a User I want to see SAP Equipment creations chronologically reported in 12NC-CV			
	Task	As a User I want to see Notification creations (MN, DN) chronologically reported in 12NC-CV			
	Task	As a User I want to see PQR changes chronologically reported in 12NC-CV			
	Task	As a User I want to see I/O spec changes chronologically reported in 12NC-CV			
7737	Story	As a User I want to see details of events related with a 12NC chronologically reported in 12NC-CV	Released	24	2017-02-21
	Task	As a User I want to see SAP status changes chronologically reported in 12NC-CV			
	Task	As a User I want to see VQI changes chronologically reported in 12NC-CV			
	Task	As a User I want to see SNP changes chronologically reported in 12NC-CV			
	Task	As a User I want to see TPD changes chronologically reported in 12NC-CV			

	Task	As a User I want to see Document changes chronologically reported in 12NC-CV			
7799	Story	As a User I want to see 12NC details frozen at the right side of the report in 12NC-CV	Released	2	2017-02-24
7800	Story	As a User I want to see 12NC details columns vertically collapsed in 12NC-CV report	Released	4	2017-02-27
7801	Story	As a User I want to export to Excel 12NC-CV reports with approximately 5000 rows	Won't fix	2	2017-03-01
7802	Story	As a User I want to see 12NC creation date and SNP reported in 12NC-CV	Released	4	2017-03-02
7803	Story	As a Developer I want to implement a new 12NC picker for 12NC-CV	Released	4	2017-03-13
7804	Story	As a Developer I want to investigate why there are not delivered equipments with notifications attached in 12NC-CV	Done	2	2017-03-09
7830	Story	As a User I want to see Activity code ZQSUPCOM for Notifications in 12NC-CV	Released	4	2017-03-08
7831	Story	As a User I want to see Detection Code Information for Notifications in 12NC-CV	Released	4	2017-03-06
7881	Story	As a User I want to reset my filters in 12NC-CV	Released	2	2017-03-16
7872	Story	As a Developer I want to add a menu tile for access 12NC-CV in SCE Horus menu	Released	2	2017-03-17
7880	Epic	12NC-Timeline Warnings			2017-04-05
7882	Story	As a User I want to see explanation columns about activity codes and TPDs in 12NC-CV	Released	4	2017-03-21
7883	Story	As a User I want to see information about the vendor of the part in 12NC-CV	Released	8	2017-03-23
7921	Story	As a Developer I want to add proper access control based on user roles to 12NC-CV	Released	4	2017-03-24
7884	Story	As a User I want to see information about the SAP equipment deliveries worldwide 12NC-CV	Released	4	2017-03-27
7888	Story	As a User I want to see the author of the SNP changes in 12NC-CV	Released	8	2017-03-29
7925	Story	As a User I want to see date related with the vendor of the part in 12NC-CV	Released	2	2017-03-30
7942	Story	As a Developer I want to investigate the inconsistency between #MNs in part info page and 12NC-CV	Done	16	2017-04-12
7941	Story	As a User I want to see PO information for SAP Equipments in 12NC-CV	Released	4	2017-04-14

7885	Story	As a User I want to see warnings based on data reported in 12NC-CV	Released	32	2017-04-21
7995	Story	As a User I want to see Bad from Stock MNs marked as failures in 12NC Timeline	Released	4	2017-04-25
7996	Story	As a Developer I want to rename 12NC-CV to 12NC Timeline	Released	4	2017-04-28
7997	Story	As a User I want to see all Notifications related with the 12NC in 12NC Timeline	Released	8	2017-05-04
8106	Story	As a Developer I want to add Warnings in 12NC Timeline via mapping instead of query left join	Won't fix	16	2017-05-09
8098	Story	As a User I want to filter out from Warning 1 those MNs linked to an Equipment ordered before the TCD	Released	8	2017-05-11
8099	Story	As a Developer I want to add TPD explanation texts in 12NC Timeline	Released	8	2017-05-12
8140	Epic	12NC Timeline - PDR			2017-06-02
8141	Story	As a User I want to have a 12NC Timeline manual	Done	16	2017-05-26
8189	Story	As a User I want to distinguish supplier related failure MNs from the others in the Verify TCD warning	Released	8	2017-05-30
8190	Story	As a User I want to see TCD Risk column in PDR	Released	24	2017-06-07
8225	Story	As a User I want to see when the TCD is missing in the Verify TCD warning	Released	8	2017-06-08
8251	Story	As a User I want to see TCD Risk column in Part Details page with a link to 12NC Timeline	Released	16	2017-06-15
8252	Story	As a User I want to see a data dump about the # of failure MNs for 12NCs without TCD	Done	4	2017-06-16
8254	Story	As a User I want to see TCD Risk information in GL report	Released	16	2017-06-28
8337	Story	As a Developer I want to include several MN detection code tables in the system	Released	16	2017-07-06
8394	Story	As a Developer I want to adjust the warnings text in 12NC Timeline	Released	8	2017-07-12
8451	Story	As a User I want to get a pre-filtered result when selecting TCD Risk in GL report	Released	8	2017-07-14
End of Project				**Total: 340h**	**2017-07-30**

A2. Data consistency

Nº	Part	SAP Equipment 12NC	In 12NC Timeline?	In PIP?	EN →	EN ←	Notification 12NC	In 12NC Timeline?	In PIP?	Explanation
1	4999.999.12345	4999.999.12345	✓	✓	10001	-	-	-	-	Correct situation where there are no Notifications attached to the SAP Equipment.
2	4999.999.12345	4999.999.12345	✓	✓	10001	10001	4022.123.12345	✓	✓	Correct situation where a Notification is attached to the SAP Equipment.
3	4999.999.12345	4999.999.12345	✓	✓	10001	10001	4999.999.99999	✓	✗	Incorrect situation where a Notification is attached to the SAP Equipment but its 12NC is not the one of the part. The Notification is showed in 12NC Timeline but not in PIP.
4	4999.999.12345	4999.999.12345	✓	✓	10001	10002 or NULL	4999.999.12345	✓	✓	Incorrect situation where a Notification is not attached to the SAP Equipment (EN different or NULL) but its 12NC is the one of the part. The Notification is showed in 12NC Timeline and PIP as an orphan Notification.
5	4999.999.12345	4999.999.11111	✗	✗	10001	10001	4999.999.12345	✓	✓	Incorrect situation where a SAP Equipment is not linked to the part but it has a Notification which is linked to the part. The Notification is showed in 12NC Timeline and PIP as an orphan Notification.
6	4999.999.12345	4999.999.11111	-	-	10005	10005	4999.999.11111	-	-	Correct situation where the SAP Equipment is linked to a different part.

Table 3: SAP Equipments and Notifications relations